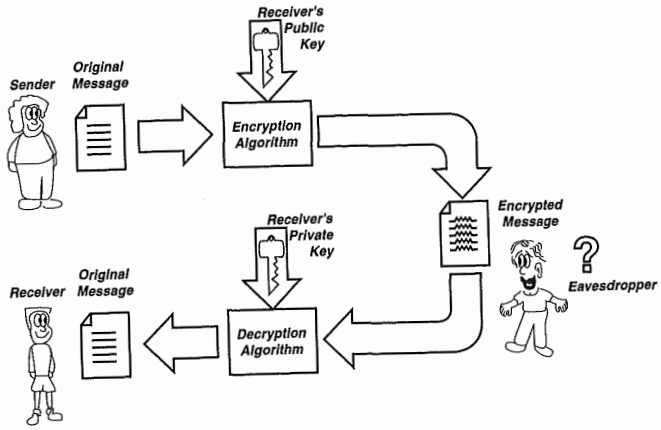
Figure 8.5

How pubic key cryptography protects privacy.



themselves, which are needed for decryption, cannot be recovered from the product. This is, therefore, an excellent framework for a public-key cryptosystem.

Comparing Secret-Key and Public-Key Cryptography

In classical secret-key cryptosystems, the encryption key gives away the decryption key. The chance that a hostile party could discover the secret key while it is being transmitted, however, does exist. In public-key cryptosystems, the encryption key can be safely publicized without compromising the secrecy of the decryption key. For this reason, classical secret-key systems also are referred to as *symmetric* or *two-way*, and public-key systems are referred to as *non-symmetric* or *one-way*.

Public-key cryptosystems are considerably slower to use than classical secret-key cryptosystems. Depending on implementation, they can be anywhere from 100 to 1,000 times slower. The best solution, however, is to combine the security of public-key systems with the speed of secret-key systems. Key management problems present in the secret-key systems can be taken care of by a suitable protocol for key exchange such as Diffie-Hellman or RSA. The public-key system can be used for secure transmission of the secret session key, which then is used to encrypt the bulk of the message.

Numerous public-key cryptosystems, based upon quite diverse concepts, exist today. The El Gamal system, for example, is based upon the discrete logarithm problem (ElGamal 1985). Cryptosystems also exist that are based upon operations on mathematical operations on elliptic curves (Miller 1986; Koblitz 1987). An advantage of the RSA public-key cryptosystem over some other public-key

IPR2025-00714
DATABRICKS EX1008 Page 209

cryptosystems is that it also can be used to construct digital signatures to authenticate documents and to ensure the integrity of document contents.

Digital Signatures

The lack of secure authentication and integrity has been a major obstacle in achieving the promise of a paperless society. Paper still is the medium of choice for all contracts, official letters, legal documents, and identification. It has not been feasible to evolve into a society based upon electronic transactions. Digital signatures hold the promise for converting all paper transactions, from the mundane to the most critical, to digital electronic media. An accepted standard for digital signatures would, for example, allow leases, wills, passports, college transcripts, and checks to exist primarily in an electronic form. Any paper version of such documents simply would be a secondary copy derived from its electronic original.

Furthermore, digital signatures have the potential to possess greater legal authority than handwritten signatures. If a 10 page contract is signed by hand on the tenth page, there is no certainty that the preceding nine pages have not been altered. If the contract has been signed by a digital signature, a third party can easily verify that not one bit of the contract has been changed.

Digital signatures can perform two important functions, namely authentication and integrity, that relate to transaction security:

- **Authentication.** A digital signature makes it possible for the receiver, as well as any interested third party, to mathematically verify the identity of the person who signed the message.
- **Integrity.** A digital signature makes it possible to detect whether the signed message has been altered after it was signed.

How Digital Signatures Work

Here is the basic outline of how the digital signature works. Suppose Alice wants to send a signed message to Bob. Alice first performs a computation that involves both her private key and the message. The result is a digital signature that is then appended to the message and sent off over the wire. Bob, to verify the signature, performs a computation involving the message, the purported message, and Alice's public key. If the result holds true in a simple mathematical equation, the signature is genuine. Otherwise, the signature may be forged or the message altered, and they are discarded. This scenario is illustrated in figure 8.6.

The generation of a digital signature requires the use of a mathematical hash function on the message. This hash function distills the information contained in a variable-size message into a fixed-size string, which acts as a "digital fingerprint" of the longer message and is called the message digest. It must be infeasible for anyone to either find a message that hashes to a given value or to find two messages that hash to the same value. Well-known secure hash functions include SHA (Secure Hash Algorithm, from NIST), MD2, MD4 and MD5 (MD for Message Digest, by Ron Rivest). The creation of a message digest is illustrated in figure 8.7.

IPR2025-00714
DATABRICKS EX1008 Page 210

Figure 8.6

Signing and verifying a message with digital signature.

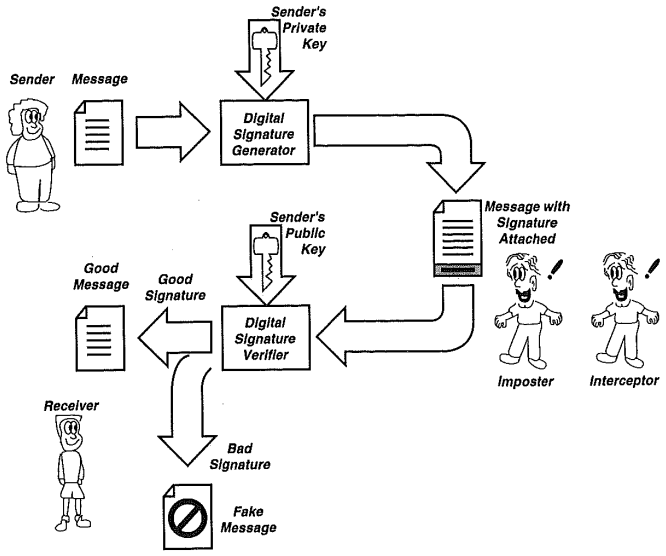
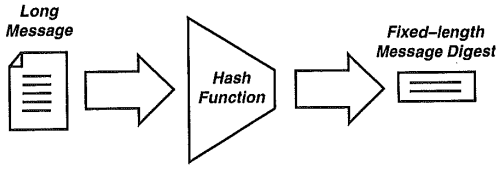


Figure 8.7

One-way hash function produces message digest from long message.



As illustrated in figure 8.8, Alice encrypts the resulting message digest with her private key to generate a digital signature, which is then appended to the message and sent off to Bob. It is more efficient to compute a digital signature using the message digest, which is small, than it is using the arbitrarily large message. After all, hash functions run faster than signing functions. In addition, a message digest can be made public without revealing the contents of the message from which it derives.

Bob, upon receiving both the message and the attached digital signature, decrypts the signature with Alice's public key to recover the message digest. He then hashes the message with the same hash function Alice used and compares the result to the

IPR2025-00714
DATABRICKS EX1008 Page 211

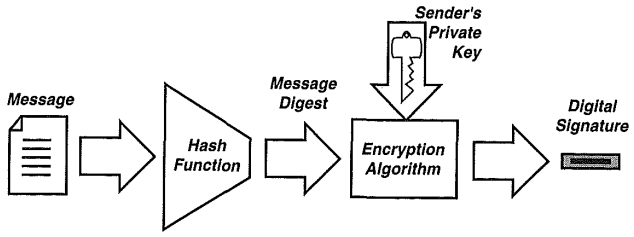


Figure 8.8

Signing a message with digital signature.

message digest decrypted from the signature. If they match exactly, the signature has been successfully verified and Bob can be confident that the message did come from Alice. If, however, the two message digests fail to match, then the message either originates from elsewhere or was altered after it was signed, and Bob rejects the message. This is illustrated in figure 8.9.



For digital signatures, the role of the public and private keys is reverse of that for message encryption. For digital signatures, the sender's private key is used to encrypt, and the public key is used to decrypt. Whereas for message privacy, the receiver's public key is used to encrypt and private key to decrypt.

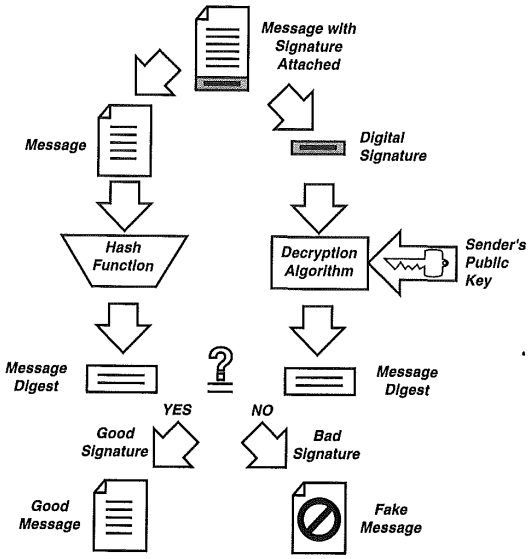


Figure 8.9

Verifying the digital signature of a message.

IPR2025-00714
DATABRICKS EX1008 Page 212

The Digital Signature Standard

Besides RSA, there are many other ways to sign a message digitally. One such scheme, called the Digital Signature Algorithm (DSA), is based upon the discrete log problem and derives from cryptosystems developed by Schnorr (1989) and ElGamal (1985). The DSA was selected by NIST, in cooperation with the NSA, to be included as part of its proposed Digital Signature Standard (DSS) for the U.S. government (NIST 1992). For various reasons both technical and political, however, DSS has received a fair number of criticisms from the computer industry. The most serious one appears to be that, unlike RSA, DSS has not been around long enough for critical examinations and, therefore, may not be secure enough (RSA 1993).

For digital signatures to replace hand-written signatures, it must have the same legal standing in a court of law as its handwritten counterpart. Although preliminary legal research thus far has resulted in the opinion that digital signatures would meet the requirements of legally binding signatures for most purposes (US 1991), its validity has never actually been tested in a court of law. Until the courts issue rulings that collectively define which digital signature methods, key sizes, and security precautions are acceptable, the surest thing to say as of the date of this writing is that the legal status of digital signatures is not yet well-defined.

Key Certification

Anyone who wants to receive encrypted messages or to sign messages must have a key pair. Some people might have more than one key. Other entities (for example, physical entities such as modems, workstations, and printers, and organizational entities (such as corporate departments, government branches, and university registrar's office) also will have keys.

To obviate the need for sending any keys at all over the network, all users should generate their own pubic/private key pairs locally. In other words, there will be no central administrative body that generates and distributes keys. Each computer on the network should be capable of generating key pairs. In this way, private keys are never transmitted across the network and no external key source need be trusted. Of course, the local key generation software must be trustworthy.

After the key pair has been generated, the public key can be widely distributed over the network. But how can one be sure that a public key found associated with a particular name really belongs to the named entity? Without a secure mechanism to safeguard the legitimacy of public keys, an imposter can easily distribute forged public keys to impersonate other users.

Public key certificates are digitally signed documents that bear witness to the fact that a given public key belongs to a given entity. They are digital documents created for the purpose of instilling confidence in the legitimacy of other people's public keys. They can help prevent an imposter from using phony public keys to impersonate others.

Certifying Authority

When a key pair is generated, the user must register his or her public key with some central administration, called a *certifying authority*. In response, the

IPR2025-00714
DATABRICKS EX1008 Page 213

certifying authority issues a certificate to the user attesting to the veracity of the user's public key, signed with its own private key. When the certificate is enclosed with signed messages, the receiver verifies the certificate by using the certifying authority's public key. Now confident of the public key of the sender, the receiver proceeds to verify the signature on the message.

A certifying authority can be any trusted central administration willing to vouch for the identities of the parties to whom it issued certificates. Since the certifying authority must check for proper identification, it is easier for an organization to vouch for its members or employees, versus unaffiliated persons. A company can issue certificates to its employees, a university to its students, and a town to its citizens. As an example, under Apple Computer's Open Collaborative Environment (AOCE), users can generate a key pair, send a notarized request, and receive a certificate for the public key.

When there are more than a handful of certifying authorities, how could one be sure of the legitimacy of a particular certifying authority's public key in the first place? The certifying authority in question could enlist the help of another, better-known certifying authority who could in turn vouch for the legitimacy of the certifying authority's public key, and hopefully in whose public key the user could be confident. If not, there may be yet another certifying authority which could in turn vouch for it, and so on. One could easily see a recursive pattern in the above logic. But ultimately the chain of certificates has to stop; there must be a trusted certifying authority at the top-level whose well-published public key everyone could be sure of.

Certificate Format

In its simplest form, a certificate contains a public key and a name. Usually, a certificate also contains the creation and expiration dates of the key, the name of the certifying authority, the serial number of the certificate, and perhaps other information as well. Most importantly, it has been signed with the certifying authority's private key and, therefore, contains the digital signature of the certificate issuer. The most widely accepted certificate format is defined by the CCITT X.509 international standard (CCITT 1988), which is further refined in the PEM (Kent 1993) and PKCS standards.

PEM is a proposed Privacy-Enhanced Mail standard for carrying secure electronic mail over the Internet (Linn 1993; Kent 1993). PEM uses a trust model that depends upon a hierarchical certificate structure. With PEM, each public key certificate must be signed by another certifying authority higher in the hierarchy, with the chain finally ending in a top-level Internet certifying authority that resides at the root. In contrast, PGP (Pretty Good Privacy, a freely available encryption program written by Phil Zimmermann) uses a decentralized model of key certification that is distributed in a "web of trust" (Garfinkel 1995).

Two Approaches to Web Security

Over the past couple of years, cryptography has taken on new commercial significance as electronic commerce and digital transactions on the Internet, and especially the World Wide Web, have increased the volume of traffic. Although currently most of

IPR2025-00714
DATABRICKS EX1008 Page 214

the transactions with remote servers are carried out directly by a human being, it is expected that in the near future, agents deployed on the Internet shall be able to transact business for their human masters electronically on the Web.

A number of security issues can arise between Web clients and servers. It's important, for example, to have privacy of information when credit card numbers are transmitted over the network. In addition, mutual authentication is frequently needed between clients and servers in order to fully support electronic commerce. Furthermore, messages need to be protected from modification during transmission.

There are a number of possible solutions, but most of the current work is concerned with securing the HTTP protocol. There are several proposed solutions to the security issues described previously, but so far there are only two leading contenders available in the commercial market:

- Secure Socket Layer (SSL) by Kipp Hickman of Netscape Communications, Inc. (Hickman 1994)
- Secure HTTP (S-HTTP) by Eric Rescorla and Allan Schiffman of Enterprise Integration Technologies Corporation (RS 1994)

No clear security standard for the World Wide Web exists at this point, although both SSL and Secure HTTP protocols utilize a variety of industry standards and protocols such as HTTP, TCP/IP, and the popular RSA cryptographic toolkits. Agents on the Web will need to understand one or the other, or perhaps both, of the SSL and Secure HTTP protocols in order to operate in the new domains of digital economy.

SSL and Secure HTTP address the problem of security from two different perspectives. Within the framework established by the seven-layer ISO/OSI protocol stack, SSL functions in the middle transport layer. It chooses to secure the channel of communication between two parties by cryptographic means to ensure that it becomes private and authenticated. SSL is simple and relatively straightforward to implement and has been widely deployed in Netscape browsers and servers.

However, Secure HTTP functions at the highest application layer and supports end-to-end secure transactions. It prefers to work with individual documents to ensure their privacy, authenticity, and integrity. Secure HTTP appears to be more flexible in that it has many hooks for providing comprehensive security through various external cryptographic methods. Digital signatures like RSA or NIST's DSS, for example, can be specified in the Secure HTTP protocol and implemented in supportive Web clients and servers.

In a broad sense, Secure HTTP can interoperate with SSL. SSL implements security at the channel or transport level. Secure HTTP, however, implements security at the application, or document level. It is therefore possible for client and server applications to negotiate a common level of security in a manner transparent to the user, and thus be able to interoperate.

SSL provides a form of transaction security that is adequate for supporting simple applications like the transfer of credit cards over the Internet. At the same time, Secure HTTP provides additional hooks for implementing a substantially richer set of security features (or offering more choices of cryptographic methods) that general purpose electronic

IPR2025-00714
DATABRICKS EX1008 Page 215

commerce might require. It is in this way anticipated that service providers will want to use SSL to get started quickly, and later develop more sophisticated applications using Secure HTTP as the need arises.

Secure Socket Layer (SSL)

Transmission Control Protocol/Internet Protocol (TCP/IP) is the official communications language of the Internet; all communicating programs must understand and use it in one form or another. It provides a reliable channel for data transport between any two communicating entities on the Internet. Many client/server applications on the Internet, such as the World Wide Web, e-mail, and network news, are written using a programming abstraction called the *sockets*, which provides access to the underlying TCP/IP protocol.

Developed by Kipp Hickman of Netscape Communications, Secure Socket Layer (SSL) is a protocol that uses a security-enhanced abstraction of sockets that takes care of transaction security at the transport level (Hickman 1994). With SSL, security properties are attached to the channel of communication that is established between the two communicating parties, but not to the documents that are being communicated. SSL is layered on top of a reliable transport protocol like TCP to ensure data integrity during transmission. The protocol allows client/server applications to communicate in private without fear of being eavesdropped. The protocol is designed so that the servers are always authenticated and clients can be optionally authenticated.

Channel Security

The SSL protocol in essence provides "channel security," which has the following three basic properties:

- **The channel is private.** Encryption is used for all messages after a simple handshake is used to define a secret key.
- **The channel is authenticated.** The server endpoint of the conversation is always authenticated, while the client endpoint is optionally authenticated.
- **The channel is reliable.** The message transport includes a message integrity check for ensuring reliability in transmission.

SSL Messaging

The SSL protocol specifies a record header and data format, as well as a protocol handshake. The SSL handshake proceeds in two major phases. The first phase is used to establish private communications. The second phase is used for client authentication.

To illustrate a scenario of SSL protocol message flow involving a client *C* and a server *S*, the following notation commonly found in the literature is used. When something is enclosed in curly braces, such as "{something} key" then "something" has been encrypted using "key."

The exchanges below are typical of what SSL does to secure a private channel without performing client authentication:

```
C→S: CLIENT-HELLO,      challenge, cipher specs
S←C: SERVER-HELLO,      connection id, server
                        certificate, cipher specs
```


IPR2025-00714
DATABRICKS EX1008 Page 216

```
C→S: CLIENT-MASTER-KEY, {master key} server-public-
                        key
C→S: CLIENT-FINISH,     {connection id} client-
                        write-key
S←C: SERVER-VERIFY,     {challenge} server-write-
                        key
S←C: SERVER-FINISH,     {new session id} server-
                        write-key
```

Each SSL endpoint uses a pair of ciphers per connection, for a total of four ciphers. At each endpoint, one cipher is used for outgoing communications and one is used for incoming communications. When the client or server generates a session key, they actually generate two keys, the server-read-key (also known as the client-write-key) and the server-write-key (also known as the client-read-key). The master key is used by the client and server to generate the various session keys.

Advantages of SSL

The advantage of the SSL protocol is that it is application protocol independent and does not depend on higher-level protocols for security. A higher-level application protocol (such as HTTP, FTP, or Telnet) can be layered on top of the SSL protocol transparently. In other words, there is no need to rewrite or to recompile the existing application programs to realize the benefit of additional security. The SSL protocol can negotiate an encryption algorithm and a session key, as well as authenticate the server and the client before the application protocol transmits or receives its first byte of data. All the application protocol data are automatically encrypted for transmission, ensuring privacy.



The SSL protocol has been submitted to the W3 Consortium (W3C) Security Group for consideration as part of a general security approach for the Web. The W3C is an organization run by Massachussetts Institute of Technology that develops common standards for the evolution of the World Wide Web. A reference implementation of the SSL protocol called *SSLRef* has been developed by Netscape Communications Inc. of Mountain View, California, and made freely available for non-commercial use.

Secure HTTP (S-HTTP)

Secure HTTP, or just S-HTTP, is a security-enhanced version of HTTP that consists of a set of protocol changes designed to address issues of privacy, authentication, and integrity (RS 1994). Secure HTTP provides transaction security at a document level where each document can be marked as private and signed by the sender. Secure HTTP's design goal is to provide a flexible protocol that supports multiple orthogonal operation modes, key management mechanisms, trust models, cryptographic algorithms, and encapsulation formats through option negotiation between the parties for each transaction.

The protocol emphasizes maximum flexibility in the choice of key management mechanisms, security policies, and cryptographic algorithms. Such application flexibility is achieved at the expense of implementation complexity. In a full-fledged implementation of Secure HTTP, quite a wide variety of cryptographic algorithms and operation modes might have to be accommodated. On the other hand, clients and servers that have chosen to

IPR2025-00714
DATABRICKS EX1008 Page 217

implement different subsets of the protocol for minimal compliance might not be able to enjoy all the benefits of secure transaction as full interoperation may be problematic.

Currently, several cryptographic message format standards can be incorporated into S-HTTP clients and servers, including, but not limited to, PKCS-7, PEM, and PGP. S-HTTP supports interoperation among a variety of implementations, and is compatible with HTTP. S-HTTP aware clients can talk to S-HTTP oblivious servers and vice versa, although such transactions obviously would not use S-HTTP security features.

Summary of S-HTTP Features

As discussed in RS 1994, the important features of Secure HTTP can be summarized as follows:

- The protocol provides symmetric capabilities to both client and server (in that equal treatment is given to both requests and replies, as well as for the preferences of both parties) while preserving the transaction model and implementation characteristics of the current HTTP.
- S-HTTP does not require client-side public key certificates (or public keys), supporting symmetric session key operation modes. This is significant because it means that spontaneous private transactions can occur without requiring individual users to have an established public key. Although S-HTTP will be able to take advantage of ubiquitous certification infrastructures, its deployment does not require it.
- S-HTTP supports end-to-end secure transactions, in contrast to the existing de facto HTTP authorization mechanism that requires the client to attempt access and be denied before the security mechanism is employed. Clients can be "primed" to initiate a secure transaction (typically using information supplied in an HTML anchor) and to support encryption of fill-out forms, for example. With S-HTTP, no sensitive data need ever be sent over the network in the clear.
- S-HTTP provides full flexibility of cryptographic algorithms, modes, and parameters. Option negotiation is used to allow clients and servers to agree on transaction modes (should the request be signed, encrypted, or both? What about the reply?); cryptographic algorithms (RSA versus DSA for signing, DES versus RC2 for encrypting, and so on); and certificate selection (please sign with your "MasterCard certificate").
- Secure HTTP has deliberately restricted itself to standardizing on message formats, independent of policy and implementation details. S-HTTP attempts to avoid presuming a particular trust model, for example, and can thus accommodate different arrangements of certifying authorities, that is, whether organized in a hierarchical fashion or in a PGP-style web of trust.

Modes of Operation in S-HTTP

Message protection in S-HTTP can be provided along three orthogonal axes: encryption, authentication, and digital signature. Any message can be encrypted, authenticated, digitally signed, or any combination of these (including unprotected).

Multiple key management mechanisms are provided in S-HTTP, including ordinary password-style shared secrets, Diffie-Hellman style public key

IPR2025-00714
DATABRICKS EX1008 Page 218

exchanges, and Kerberos (SNS 1988) ticket distribution. In particular, provisions have been made for prearranged (in an earlier transaction) session keys to send confidential messages to those who have no key pair.

Secure HTTP provides a means to verify sender authenticity and message integrity for a HTTP message via the computation of a Message Authentication Code (MAC). This mechanism also is useful in cases where it is desirable to allow parties to identify each other reliably in a transaction.

If the digital signature enhancement is adopted, an appropriate certificate might be attached to the message (possibly along with a certificate chain) or the sender might expect the recipient to obtain the required certificate (chain) independently.

The protocol provides a simple challenge-response mechanism, called *nonce*, allowing both parties to assure themselves of transaction freshness. Additionally, the integrity protection provided to HTTP headers permits implementations to consider the "Date:" header allowable in HTTP messages as a freshness indicator, where appropriate.



S-HTTP has been submitted to the Internet Engineering Task Force (IETF) working group on Web transaction security for consideration as an Internet standard. The IETF is a standards body consisting of a self-organized group of people that makes technical and other contributions to the engineering and evolution of Internet and its technologies. Reference implementations of Secure HTTP in the form of Secure NCSA Mosaic and Secure NCSA httpd are available from Enterprise Integration Technologies (EIT) of Menlo Park, California.

Current Practice and Future Trend in Web Security

Transaction security is vital to the growth of the emerging information infrastructure and the widespread deployment of agents in the commercial domains. There is no clear security standard at this point for the World Wide Web. A concern in the market is the development of two different security approaches, SSL and Secure HTTP, that cannot communicate with each other. Information providers must be concerned with how easily information servers can interoperate with clients on the Web.

The W3C, IETF, and other relevant standards groups recognize the importance of security on the Web. In due course, a common security standard will be defined by the standards groups. The formal process of standardization, however, is lengthy and could run for years. In the interim, it is widely recognized that a common interoperable approach to transaction security should make it easier for information providers to provide secure information on the World Wide Web, and make it easier for consumers to access it.

Such a universal approach currently is being provided by Terisa Systems of Menlo Park, a joint-venture between EIT and RSA Data Security, which offers a set of developer's toolkits that combine both SSL and Secure HTTP. By following a unified approach that allows information providers to take advantage of the strengths of both protocols, developers can be assured that their applications will interoperate securely with other applications deployed on the Web while they wait for a formal standard to become established and widely adopted.

IPR2025-00714
DATABRICKS EX1008 Page 219

In July 1995, Open Market Systems, based in Cambridge, Massachusetts, released its Secure WebServer product and became the first company to field a Web server that can handle both of today's dominant security schemes: SSL and Secure HTTP. The convergence toward a common approach could mean that merchant applications that depend on transaction security, such as payment, inventory, and delivery systems, will be made available much more rapidly because developers will find it far more attractive to create Web products and Web commerce agents using a universal security approach.

IPR2025-00714
DATABRICKS EX1008 Page 220

chapter 9

Electronic Cash and Payment Services

The advent of electronic commerce on the Internet is to a large extent facilitated by the launch of the World Wide Web. In the realm of agents, there are currently hordes of spiders, wanderers, brokers, and bots on the Web performing various tasks for their human clients: for example, searching for information, maintaining Web infostructure, brokering for buyers and sellers, as well as finding the best bargain of books and CDs online.

IPR2025-00714
DATABRICKS EX1008 Page 221

Digital cash, a string of numbers made up of zeroes and ones, is the electronic equivalent of cash on the Internet. Instead of being engraved on special paper with watermarks to prevent counterfeiting, digital cash is protected from forgery by strong cryptography and can be stored on computer hard disks or transported over the network. Similarly, electronic versions of different paper-based payment schemes have also been invented for the Internet.

Agents that are active in the online marketplace must be able to handle money and payments in an electronic form for full-blown electronic commerce. Examples of online tasks that could be automated using agents include the ordering of parts from an online catalog, the sale of information over the Internet, as well as the reservation and purchase of opera tickets on the Web.

Brief History of Money

Money came about as a major innovation in the early economic systems to serve as a medium of exchange and a store of value. Precious metals, which were first used around 2,500 B.C. in Mesopotamia, were an early form of money. In U.S. history, tobacco was used in Colonial Virginia as a medium of exchange, and even whiskey attracted a following among frontiersmen.

Coins stamped from precious metals, which date back to 700 B.C., represent a significant advance over non-uniform pieces of metal and are preferred for their convenient form, as well as standardized sizes and content. In general, the face value of a coin exceeds the worth of the metal it contains. This difference generally is considered a fair price to pay for the added convenience that coins offer. Over time, this difference, known as *seigniorage,* has been a reliable source of revenue for mints, which tend to be government monopolies.

When paper money came along around the 11th century in China, it represented a vast improvement. Not only is paper money cheaper to produce and easier to carry, but it costs less in real terms. Challenges abound, however. From an institutional viewpoint, the issuing authority must inspire public confidence, as it is relatively easy to debase the currency by printing too much money. From a technology viewpoint, the official engravers constantly must stay ahead of counterfeiters.

In the age of electronic commerce on Internet, money has evolved into a digital form consisting of strings of zeroes and ones that can be stored on disks for safekeeping and can zap across networks in seconds during payments. There are myriad entrepreneurial startup companies professing to be electronic banks offering various types of digital cash, coupons, and payment services to the general Internet populace. There is DigiCash, NetBank, CyberCash, and First Virtual to mention just a few.

This chapter explores the offerings of DigiCash, CyberCash, and First Virtual in detail. For the purpose of motivating the use of digital cash and payment services, the following section examines various methods of paying for goods and services on the Internet, along with an assessment of risks involved.

IPR2025-00714
DATABRICKS EX1008 Page 222

Choice of Payment Methods

Traditionally, payment for goods and services on the Internet requires the mailing of a check. Payment generally takes longer due to the "snail pace" of the Postal Service as well as the time required to "clear" checks at the banks. Furthermore, for one-time users who must pay for small amounts of data from many diverse sources on the Internet, this payment mechanism is cumbersome (imagine having to write and mail checks to hundreds of information providers each month!).

Credit card payments on the Internet avoid the Postal Service and interbank check processings. There are, however, substantial risks involved when credit card information is sent over the Internet. Messages often pass through multiple network hosts and routers on their way to the merchant. Therefore, it is possible that somewhere along the way, someone could be scanning these messages for credit card details and compromising the security of this method.

Credit card payments on the Internet are made secure when credit card details are protected by strong encryption. The cost of processing these types of credit card transactions, however, can outweigh the benefit. Transactions worth a few dollars or less, such as the fee for a movie review or an encyclopedia entry, generally are not worth the cost of processing a credit card order.

One economical solution to small value payments on the Internet would be the introduction of a third party; that is, a company that collects and approves payments from a client to a merchant electronically. After a certain period of time has elapsed, or when a certain value threshold is reached, one single credit card transaction for the total accumulated amount would be made. For example, at the end of every month, or whenever cumulative charges for a client exceeds the $100 threshold, the client's credit card will be billed for whatever amount is outstanding.

However, there are limitations and shortcomings to this method. The possibility that a payment will be refused because the spending limit has been reached does exist. But more importantly, one's payment details, such as where you buy, when you buy, and sometimes what you buy, are gathered in a centralized database. In the wrong hands, the collection of this data can pose a serious transgression on one's personal rights to privacy.

In real life, people can pay with cash, anonymously if desired. To introduce this same facility on the Internet, one would need the electronic equivalent of cash on the Internet, called *digital cash*.

What is Digital Cash?

Digital cash is the electronic equivalent of cash on the Internet. It is a string of zeroes and ones whose pattern represents money that cannot be forged. You can withdraw "digital coins" from an Internet bank account and store them on local hard disks. You then could use these coins for making payments electronically. Digital cash can be easily transported from one place to another over the Internet. All payments are fast, convenient and, more importantly, anonymous. Furthermore, you can always prove that you have made certain payments.

IPR2025-00714
DATABRICKS EX1008 Page 223

Digital cash offers speed, convenience, anonymity, and proof of payments. Another desirable property of digital cash, which is significantly harder to achieve, is that the payer should be able to make offline payments to arbitrary payees in such a way that the latter can verify the validity of payment (that is, no double spending of the same note) without using the network. Digital cash systems with this property are called *offline digital cash*, and are highly complex. Details can be found in Chaum 1992 and Brands 1993, 1994.

Digital cash can be used for payments that are too small to cost-effectively process as individual credit card or debit card payments. By offering the benefits of speed and hands-free automation, digital cash can, for example, find uses in paying for small charges that result from scanning a movie review or looking up an encyclopedia entry.

A system for processing digital cash payments is key to unleashing the projected explosion in entrepreneurial electronic information publishing. The traditional way of handling database retrievals through pre-arranged subscriptions is no longer workable in the new era of spontaneous electronic commerce on the Internet. There are simply too many independent information services and unaffiliated database publishers to subscribe to on a one-to-one basis. DigiCash, a company based in Amsterdam, the Netherlands, is currently developing, refining, and field-testing a digital cash system for use on the Internet. I will have more to say on this in a later section of this chapter.

Digital Cashier's Check

As illustrated in Chapter 8, digital signatures can be used to provide all manners of unforgeable credentials, as well as other services. Now consider how digital signatures might be used to provide an electronic replacement for cash. The following scenario is excerpted from David Chaum's very well-written article in *Scientific American* (1992). David Chaum is the head of the Cryptography Group at the Center of Mathematics and Computer Science (CWI) in Amsterdam, as well as the founder in 1990 of DigiCash, which is discussed later in this chapter.

Suppose that a fictitious First Digital Bank offers electronic bank notes, which are messages signed using a particular private key of the bank. All messages bearing one key might be worth a dollar, all those bearing a different key five dollars, and so on, for whatever denominations are needed.

These electronic bank notes could be authenticated using the corresponding public key, which the bank has published as a matter of public record. For purpose of correspondence, First Digital would also make public a key to authenticate electronic documents sent from the bank to its customers.

To withdraw a dollar from the bank, Alice first has to generate a note number. Each note bears a different number, akin to the serial number on a bill. Suppose that she picks a 128-digit number generated at random so that the chance anyone else would generate the same one is negligible. She signs the number with the private key corresponding to the public key that she has previously established for use with her account.

IPR2025-00714
DATABRICKS EX1008 Page 224

The bank verifies Alice's signature and removes it from the note number, signs the note number with its worth-one-dollar signature and debits her account. The bank then returns the signed note along with a digitally signed withdrawal receipt for Alice's records.

In practice, the creation, signing, and transfer of note numbers would be carried out on Alice's desktop computer. The power of the cryptographic protocols, however, lies in the fact that they are secure regardless of physical medium; the same transactions could be carried out using only pencil and paper.

When it comes time for Alice to pay for a purchase at Bob's shop in cyberspace, she transfers one of the signed note numbers the bank has issued to her. After verifying the bank's digital signature, Bob transmits the note to the bank, much as a merchant verifies a credit card transaction today.

The bank reverifies its signature, checks the note against a list of those already spent, and credits Bob's account. It then transmits a "deposit slip," once again unforgeably signed with the appropriate key. Bob hands the merchandise to Alice, along with his own digitally signed receipt, completing the transaction.

This system provides security for all three parties. The signatures at each stage prevent any one from cheating either of the others—the shop cannot deny that it received payment, the bank cannot deny that it issued the notes or that it accepted them from the shop for deposit, and the customer can neither deny withdrawing the notes from her account nor can she spend them twice.

Privacy Issues

Such electronic bank notes just described are more like digital cashier's checks than digital cash. According to Chaum, this system is secure but has no privacy. If the bank keeps track of note numbers, it can link each shop's deposit to the corresponding withdrawal and determine precisely where and when Alice spends her money. The resulting dossier is far more intrusive than those now being compiled.

Furthermore, records based on digital signatures are more vulnerable to abuse than conventional file records because they are self-authenticating. In other words, even if these self-authenticating records are copied, the information they contain can be verified by anyone. In contrast, no simple claims of authenticity can be established for copies of conventional file records as they may be subject to tampering.

In addition, records based on digital signatures permit a person who has a particular type of information to prove its existence without giving the information away or revealing its source. The idea behind such zero-knowledge proofs is essentially that everything provable could be done while revealing "zero knowledge" (BGGHKMR 1988; BC 1990; Schneier 1994). Someone might, for example, be able to prove incontrovertibly that Alice had telephoned Bob on 12 separate occasions without having to reveal the time and place of any of the calls.

To restore privacy to monetary transactions in cyberspace, Chaum has invented *anonymous digital cash*, which is made possible using a cryptographic technique called *blind signatures*.

IPR2025-00714
DATABRICKS EX1008 Page 225

Anonymous Digital Cash through Blind Signatures

The idea behind anonymous digital cash is simply that digital cash transactions should remain anonymous, just as in plain cash. Chaum has developed an extension of digital signatures, called *blind signatures* (1982, 1985), which is used to implement *anonymous digital cash*. Here is how it works.

Before sending a note number to the bank for signing, Alice, in essence, multiplies it by a random factor known only to herself, and signs the resulting product with her digital signature. The note is considered "blinded" by the introduction of the random factor, also known as the *blinding factor*. As before, the bank duly verifies Alice's digital signature and removes it from the blinded note number. The bank signs the blinded note with a properly denominated signature and debits Alice's account accordingly. The bank knows nothing about what note number it is signing. It doesn't, and shouldn't, care. All the bank needs to know is that the blinded note carries Alice's digital signature.

After getting back the blinded note signed by the bank, Alice *unblinds* the note by arithmetically dividing out the blinding factor from the bank-endorsed blinded note. The beauty of the mathematics behind blinded signatures is that the resulting quotient is now the original note number with the bank's digital signature over it, without the blinding factor. The net effect is as though the bank has signed on the original note (before introducing the blinding factor) blind-folded! Alice can then spend the bank-endorsed note at any shops that accept digital cash, as before.

Such bank-endorsed notes have many properties associated with plain cash. Like cash, they can be spent at a cybershop anonymously. Furthermore, the bank notes are a negotiable instrument; that is, they are transferable from one person to another by being delivered with or without endorsement so that title ownership passes to the transferee. In other words, the holder of the bank notes owns them.

Unconditional Untraceability

According to Chaum, the blinded note numbers are "unconditionally untraceable," that is, even if the shop and the bank collude, they cannot determine who spent which notes. Because the bank has no idea of the blinding factor, it has no way of linking the note numbers that Bob deposits with Alice's withdrawals.

Whereas the security of digital signatures is dependent on the difficulty of particular computations, the anonymity of blinded notes is limited only by the unpredictability of Alice's random blinding factors. If she wants, however, Alice can reveal these random numbers and permit the notes to be stopped or traced.

Double Spending

Blinded electronic bank notes protect individual privacy. But because each note is simply a number, it can be copied easily. To prevent double spending, each note received by a merchant must be checked online against a central list of spent notes maintained by the issuing digital bank. Such a verification procedure might be acceptable when large amounts of money are at stake, but it is far too expensive to use when someone is just buying an online newspaper. To solve this problem, several offline digital

IPR2025-00714
DATABRICKS EX1008 Page 226

cash schemes have been invented, including one that is based upon the discrete log based signatures (Brands 1993, 1994) as well as one by Chaum and colleagues, which for expository purposes is conceptually easier to understand (1992).

The scheme of Chaum et. al. (1992) works as follows. Before accepting an offline payment, the payee's computer issues an unpredictable challenge to which the payer's computer must respond with some information about the note number. This information alone discloses nothing about the payer, and is collected by the bank when the note is finally deposited with the bank by the payee. Should the payer spend the note a second time, the information yielded by the next challenge, when combined with previously collected information kept at the bank, gives away the payer's identity when the note is ultimately deposited with the bank a second time. In this way, should the bank find out that the same note has been twice spent, it can proceed to identify the culprit payer by combining the two pieces of information together.

Ecash from DigiCash

Ecash is offered by DigiCash, a company founded in 1990 by David Chaum and headquartered in Amsterdam, Holland. The DigiCash home page is shown in figure 9.1. Everyone using Ecash must have an Ecash account at a digital bank on the Internet. The procedure for opening an Ecash account with a digital bank is very much the same as that of opening an ordinary checking account with a traditional bank; real money must first be deposited. Using that account, people can expect to withdraw and deposit Ecash in addition to transacting with real cash. However, real bankers are certainly a conservative lot. As of this writing, there have not been any operational digital banks on the Internet that would offer this service interconverting between Ecash and real money.

Figure 9.1

The DigiCash home page.



IPR2025-00714
DATABRICKS EX1008 Page 227

Ecash is a coin-based system, where coins have predetermined values. Every coin is a digital signature from the bank. Coins of different values are combined into the amount required for payment, just like real coins.

Starting in October 1994, DigiCash has been running an Ecash trial on a continual basis, using a fictitious DigiCash-run First Digital Bank as the trial bank. The Ecash trial has thus far attracted more than 5,000 participants on the Internet. The currency used in the Ecash trial is a virtual one called *Cyberbucks*, which does not have any relationship with real currencies. This means that cyberbucks cannot be exchanged for real money. Each participant in the trial receives $100.00 cyberbucks to be spent on goods and services ranging from computer images, software, and games to lotteries, charities, and casinos. There are more than a hundred such Ecash-accepting cybershops on Internet, a complete list of which can be found at `http://www.digicash.com/shops/alpha.html`. As shown in figure 9.2, the following steps describe the flow of Ecash in a digital economy:

1. When an Ecash withdrawal is made, the computer of the Ecash user calculates how many digital coins of what denominations are needed to withdraw the requested amount. Next, random serial numbers for those coins are generated and the blinding factors applied. The result of these calculations are then sent to the digital bank.
2. The bank digitally signs the blinded numbers with its secret key. At the same time, the client account is debited for the same amount. The authenticated coins are sent back to the user. The user arithmetically divides out the blinding factor that he introduced earlier by simple arithmetic multiplication. The serial numbers plus their signatures are now digital coins, with their value guaranteed by the bank.
3. The coins can be stored locally on the user's computer. As soon as a payment is to be made, the user's computer collects the coins needed to reach the requested total value. These coins then are sent to the receiver.
4. The receiver forwards the coins directly to the digital bank. The bank verifies the coins' validity and that they have not been spent before. To prevent double spending, there must not be more than one copy of the same bank note in circulation on the Internet. The account of the receiver is credited.

   Every coin is used only once. Another withdrawal is needed if the receiver wants to have new coins to spend.

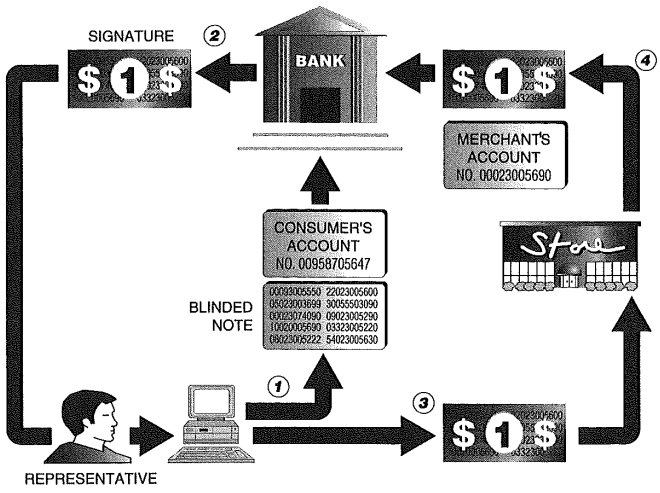


Like travelers checks, Ecash can handle lost money very well. Ecash stored on disk can be lost in a disk crash if not backed up onto tape, for example. Lost Ecash notes can be reported to the bank along with the serial numbers on the lost coins. These serial numbers allow the bank to check if the coins are really lost (or have already been spent) and refund the money accordingly.

IPR2025-00714
DATABRICKS EX1008 Page 228

Figure 9.2

Flow of Ecash in digital economy.



Ecash Security and Other Issues

Ecash offers payer anonymity based upon RSA public-key cryptography and blinded signatures, and not on the revocable policy of a remote institution. When paying with Ecash, the identity of the payer is never revealed. Each Ecash coin has its own note number (randomly generated by the user and not by the bank) that has been blinded by the user before presentation to the bank for signing. The bank has no way to find what note numbers it has signed.

Neither the payer nor the payee can counterfeit the bank's signature. Both can verify that the payment is valid because each has the bank's public key. The payer can prove that payment has been made because he can make available the blinding factor. But because the original note number was blinded when it was signed, the bank cannot connect the signing with the payment. All three parties are thus protected: the bank is protected against forgery, the payee against the bank's refusal to honor a legitimate note, and the payer against false accusations of non-payment and invasion of privacy.

Safeguards against Crime

Ample safeguard is built into Ecash against criminal use of anonymity in digital cash. What Ecash offers is more accurately described as one-side anonymity; that is, only the payer is anonymous, the recipient of the money has no anonymity at all.

All money that the payee receives must be cleared with the bank. Ecash banks are thus able to list the amounts of all payments received for all accounts. It is not possible to hide from the bank the fact that money has been received. As a result, tax-evasion, extortion, and bribes through Ecash is not possible.

The legal risks to black-marketeering with Ecash are even more insurmountable. Customers cooperating with the authorities can establish that a certain payment has been made to the black

IPR2025-00714
DATABRICKS EX1008 Page 229

marketeer, retroactively. As DigiCash puts it succinctly, "Paper cash will remain the criminal's favorite."

Payment Systems on the Internet

A hundred years ago, if a local merchant in San Francisco, California wanted to buy goods from Rochester, New York, he placed his order by mail or telegraph and then arranged for his local bank to handle the payment. The banker would face the cumbersome task of finding an institution in Rochester that was part of a common clearinghouse, probably in New York City. After debiting the customer's account, the local bank would wire the New York clearinghouse and direct it to transfer the funds from its own account to that of the Rochester bank. The Rochester bank would in turn pay the supplier. The customer would even be charged for changing San Francisco dollars to New York dollars. Incredible as it might seem today, newspapers at the time published tables of domestic exchange rates (GS 1979).

U.S. Payment Systems Today

With today's technology, the same transaction now can be completed by phone, fax, or e-mail. In the United States, payments can be made by the following five types of non-electronic instruments:

- Cash
- Checks
- Credit cards
- Traveler's checks
- Money orders

Payments also can be made by the following four types of electronic instruments:

- Automated clearinghouse
- Wire transfer
- Point of sale
- Automated teller machines

Each of these instruments, however, has different production and processing costs, is subject to varying amounts of float (that is, the time between a payer liquidating his obligation and the payee receiving use of the funds), and offers different levels of convenience and security. For example, when vast amounts of money must be moved, wire transfers (Fedwire and the private Clearing House Interbank Payment System, or CHIPS) are the preferred method of payment because the funds are available almost immediately (Bauer 1994). Most transactions are relatively painless unless something goes awry, such as a check being returned for insufficient funds. The apparent ease of making payments is the result of many hidden technological and institutional arrangements.

Technology-Driven Evolution

New technologies and institutional arrangements require continual reworking of the payment system. An effective way to boost productivity is to be technology-driven, always striving to increase the productivity of payment instruments wherever possible. A recently completed, all-electronic automated clearinghouse system, for example, eliminates some volume of tapes and disks, which are expensive to process. Another example is the adoption of electronic imaging technology for paper check processing.

IPR2025-00714
DATABRICKS EX1008 Page 230

The banking industry expects that the logical next phase in the evolution of the payment system is to conduct each step of a transaction (for example, ordering, payment, and inventory) electronically through an automated clearinghouse (ACH) or through financial electronic data interchange (EDI). Financial EDI can easily replace paper checks with a less expensive ACH transaction, which could form the basis of an Internet payment service. This is an area where a new generation of agents can be deployed. Potential benefits include better control of the timing of payments and the receipts of funds, more accurate record keeping, and lower costs.

On a different technological frontier, the Internet appears to be gathering momentum in trying to shape electronic payment systems for the masses. As it stands now, serious institutional barriers to electronic commerce remain. What is needed on the Internet now is a universally acceptable, safe, secure, convenient, and immediate system for processing payments for both goods and services.

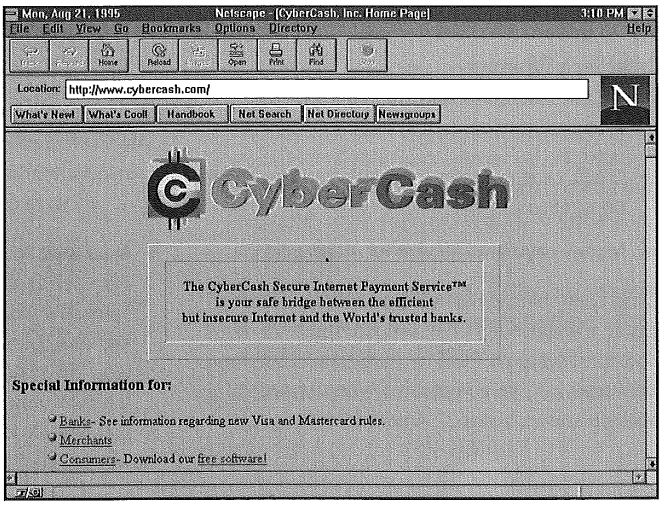
CyberCash Internet Payment Service

CyberCash delivers a safe, convenient, and inexpensive system for making payments on the Internet through partnerships with financial institutions and providers of goods and services. The CyberCash approach is based on establishing a trusted link between the seemingly unpredictable world of cyberspace and the traditional banking world.

CyberCash, based in Reston, Virginia, was founded by Bill Melton and Dan Lynch (see fig. 9.3). Melton also helped found the successful VeriFone company, which sells credit card transaction processing equipment. Lynch is founder of the popular NetWorld and Interop networking show and exhibition.

Figure 9.3

The CyberCash home page.



IPR2025-00714
DATABRICKS EX1008 Page 231

CyberCash considers itself a conduit through which payments can be transported safely and instantaneously between buyers, sellers, and their banks. Much as it is in the real world of commerce, the buyer and seller need not have any prior existing relationship. CyberCash is a neutral third party whose sole concern is ensuring the efficient, safe, and low-cost delivery of payments from one party to another, sort of like the FedEx of payments on the Internet.

For both merchants and banks, the guarantee of safe credit and debit card transactions significantly reduces the risk of fraud normally associated with these types of transactions. For merchants especially, safe credit and debit card transactions have the potential to reduce the high cost of credit card transactions via telephone or mail. Normally, bankcard associations consider these "Card Not Present" transactions riskier and more expensive and consequently charge merchants a higher processing fee.

CyberCash Offerings

CyberCash offers the following two payment services on the Internet:

- Credit card services for authorized merchant transactions
- Money payments service for individuals

CyberCash charges banks/processors fees for merchant transactions that are competitive with traditional systems. Fees for transactions between individuals will be competitive with the cost of a postage stamp, about 32 American cents.

To gain access to CyberCash services, a personal computer with a network connection is needed. CyberCash offers free client software that directly communicates with CyberCash servers, which in turn are linked to the private networks of the banks. Free merchant server software enables Internet merchants to get immediate online credit card authorizations and includes all the functions necessary to process transactions and settle with their banks.

CyberCash software provides the needed electronic link between consumers, merchants, and their banks. To facilitate access to the CyberCash system, CyberCash icons are incorporated into the graphical user interfaces of application software. Consumers using these software products can then quickly and easily enter the CyberCash system by clicking on the icon when they are ready to make payments for goods and services. Besides the normal order of commerce, CyberCash also provides service to individuals making monetary transactions. Buying a used microwave oven advertised on the net from someone, for example, and paying him through CyberCash.

CyberCash promises to "materially reduce the risk of financial transactions on the Internet" by carrying out transactions with the following properties:

- **Automated.** After the consumer enters appropriate information into the computer, no manual steps are required to process authorization or settlement transactions through the entire system. The consumer need only initiate payment for each transaction by exercising the pay option on a CyberCash form.

IPR2025-00714
DATABRICKS EX1008 Page 232

- **Safe.** Transactions are protected from both tampering and modification by eavesdroppers on the net. Cryptographic protection is used in all transmissions.
- **Private.** Information about the consumer not essential to the transaction is not visible to the merchant.

How Secure is CyberCash?

The CyberCash system is secured by strong crytography and operates on top of any general security system such as SSL or Secure HTTP.



The CyberCash system currently uses full 768-bit RSA as well as 56-bit DES encryption of messages. All transactions are authenticated with MD-5 and 768-bit RSA signatures.

Credit Card Services

Consumers need not have any prior relationship with CyberCash to use the CyberCash Credit Card Services. As illustrated in figure 9.4, the process for authorized merchant transactions is quite straightforward, consisting of the following sequence of steps:

1. After desired merchandise has been selected, the consumer clicks on the CyberCash icon when ready to pay, selects from already set-up credit cards, then clicks on the "Pay" button. Figure 9.5 shows a typical scenario involving the CyberCash user interface as seen by the consumer.
2. The merchant sends the consumer an online invoice detailing the purchase information (for example, order number, description of items, part numbers, quantities and prices, and so on) together with a CyberCash payment statement confirming the total charges.
3. When the CyberCash statement appears on the screen, the consumer enters his or her credit card number, name as it appears on the card, and an expiration date. The customer also has the choice of using a debit service to make the purchase. In this case, the customer simply adds the personal identification number (PIN) to the merchant's online invoice.
4. This statement information is then automatically encrypted and passed along with the invoice to the merchant.
5. The merchant's computer adds identification information and forwards the transaction over the Internet to the CyberCash server. The merchant cannot see the consumer's credit card number; it can only sees the information necessary to process the payment request.
6. CyberCash performs error checking and forwards all the information to the traditional, trusted credit card authorization and settlement networks. CyberCash retains no information related to the merchandise being ordered (for example, order number, description of items, part numbers, and so forth).
7. To complete the transaction, CyberCash returns an electronic receipt with credit card authorization data to the merchant.
8. The merchant then finalizes the transaction with the consumer.

IPR2025-00714
DATABRICKS EX1008 Page 233

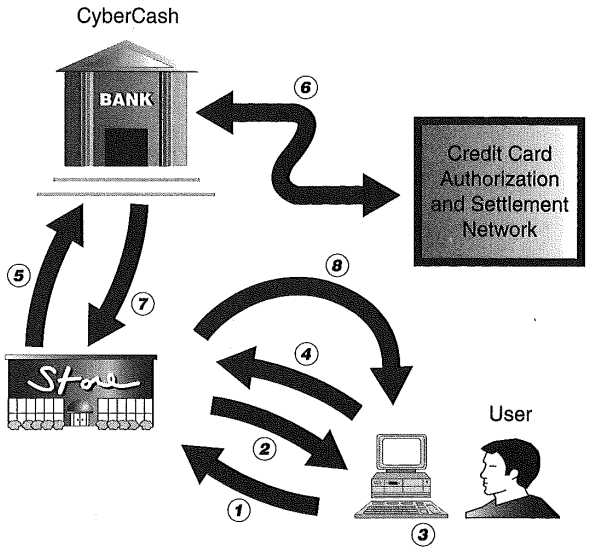


Figure 9.4

CyberCash Credit Card Services.

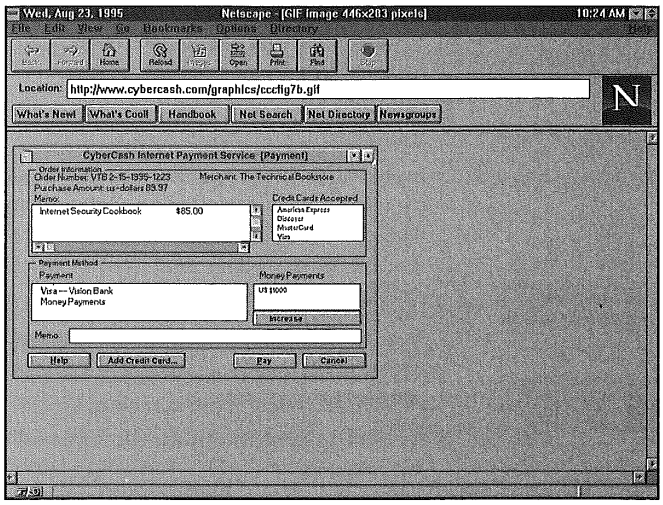


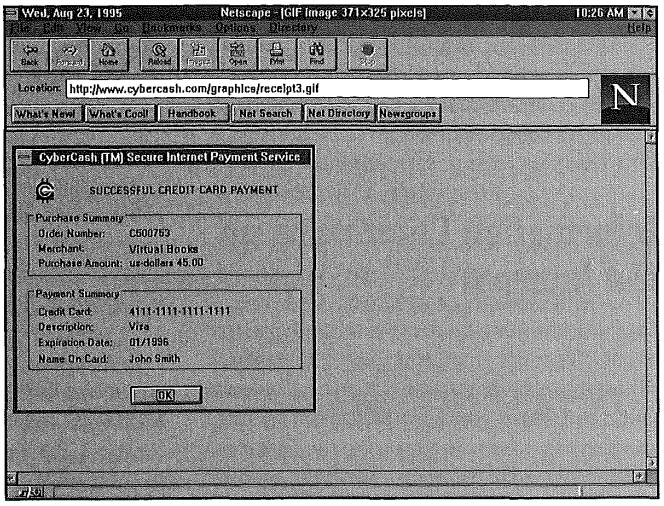
Figure 9.5

The user initiates credit card payment through CyberCash Credit Card Services.

IPR2025-00714
DATABRICKS EX1008 Page 234

Figure 9.6

Acknowledgment to user of a successful credit card payment through CyberCash Credit Card Services.



Money Payments Service

The CyberCash Money Payments Service promises to make small financial transactions on the Internet feasible and cost effective. It provides the key to enabling small payments on the Internet for items such as publications and database searches. This will provide ample revenue source for a whole new generation of information agents—that is, agents that publish, buy, sell, and trade in information—to thrive on the Internet.

Unlike CyberCash Credit Card Services, there is no clear distinction between the consumer and the merchant in the Money Payments Service. In fact, electronic cash can be transferred from any CyberCash account holder to any other Internet user with an e-mail address, whether or not they have a CyberCash account.

Individuals or organizations can establish one or more CyberCash accounts directly with CyberCash and accounts are maintained by CyberCash on the basis of an account holder's CyberCash key and not on direct user identity. Thus, an individual can have several unrelated CyberCash accounts or share a CyberCash account with other individuals, as long as they share a CyberCash key. This approach provides a degree of user privacy.

CyberCash accounts are non–interest-bearing holding accounts for cash that the account holder intends to transfer or has received through CyberCash. A Money Payment begins with a simple command to the Secure Internet Payment Service. Payments are peer-to-peer and virtually instantaneous. A cryptographically signed receipt confirms the transaction.

IPR2025-00714
DATABRICKS EX1008 Page 235

CyberCash Money Payment funds are comprised of real money kept only within real banks and are moved exclusively by them. The only way to place cash into or remove cash from a CyberCash account is through a demand deposit account with the bank.

Information Commerce on the Internet

For more than a decade, the Internet has been used to get information from the people who have it to the people who want and need it. But in the beginning, the Internet was used only for government research information. As more and more institutions became connected, scientists and other academics began using the Internet to exchange data and research results instantly, and companies began using the Internet for rapid business communications. Over the years, an Internet culture took hold, based on mutual respect, broad access, and the free and open exchange of information.

As the Internet community has exploded in size, people have begun creating and distributing fascinating types of information over the Internet that no one would have predicted 20 years ago. Everything that is on the Internet now has been put there by people gratis.

Compensation really has not been possible because there is no simple, reliable way for people to pay for the information they retrieve from the Internet in a way that respects the Internet way of doing things—browsing the net for items of interest before committing to paying for them. To be properly compensated, online information providers have adopted various charging models, including periodic subscription fees, connect-time charges, or volume charges by byte count.

With such charging models, however, it is often difficult to know whether an information product is worth purchasing unless the prospective buyer is able to examine or use it. Buyers find themselves increasingly unwilling to commit to pay for information they have not been able to see or evaluate. In the area of software, this has led to the creation of Internet-posted shareware, in which payment is encouraged only after the product has been obtained and evaluated. Internet-based shareware, characterized by free distribution and voluntary payment, was the first new economic model to emerge for information commerce.

Economics of Information Commerce

Traditionally, only two types of commerce have been possible: the sale of goods and the sale of services. Before the advent of commercial online services, information historically was sold bundled into the sale of other goods or services. Books, for example, contain information, but the nature of printing and the distribution of printed material inevitably converts the information into a physical good. Similarly, legal or medical information is sold but requires a service provider (that is, lawyer or doctor) to deliver the information.

Because information is most often sold today as part of a physical object like a book, it has associated costs of manufacturing, inventory, and distribution. Books, for example, are subject to the costs

IPR2025-00714
DATABRICKS EX1008 Page 236

of shipping, handling, trucking, warehousing, returned goods, damaged merchandise, unsold inventory, and so on. In many cases, the distribution cost of information-based "packaged goods" greatly exceeds the cost of producing the information in the first place. An author's royalty, for instance, is only a modest percentage of the total cost of a book.

With the emergence of the global Internet as an inexpensive means of global information transport, it has become possible to rapidly deliver information in electronic form, without physical packaging and for virtually no cost. After all, the cost of warehousing is nil. Duplication costs for information products are virtually zero. A copy of the product can be shipped instantly over the Internet whenever the customer makes a purchase, without incurring the traditional costs associated with physical distribution.

The costs to the seller associated with selling information over the Internet are much lower than the cost of doing business the traditional way. Furthermore, order processing and fulfillment can be entirely automated using agents, eliminating the need for manual intervention. And there is no need for returned goods; it costs more to receive returned goods than for the unsatisfied customer to simply discard them!

Agents for Information Commerce

The implications for agents are immense. At such a low threshold of cost for the sale and distribution of information, there are ample opportunities for information agents of all kinds to operate in such a virtual environment. We have previously examined in Chapter 4, "Spiders for Indexing the Web," how Web spiders have been successful in collecting and indexing Web documents in preparation for consumption by human users. Agents can also play the role of for-profit information brokers who search for, compile, buy, sell, and trade in information in such a global information marketplace now made possible by First Virtual.

As opposed to conventional commerce in goods and services, pure information commerce is special because no scarce objects change hands, and no valuable time is expended on any individual transaction. The primary risks of traditional commerce—notably the lack of payment for the delivery of a physical object or the expenditure of nonrenewable time—simply do not apply to information commerce.

Reconsideration of Risk Factors

Traditional payment mechanisms seek to establish a very strong link between the delivery of a good or service and the guarantee of payment. Thus, a merchant generally will not allow you to leave the store with the goods in your hand until he has received cash from you or until your credit card has been validated by a trusted entity, such as the Visa or Mastercard networks. This linkage makes sense for physical goods because the merchant's greatest risk is the loss of the physical object without appropriate payment. Good business practice generally implies that the merchant will, under appropriate circumstances, refund the buyer's money upon the return of the physical object.

The risk factors are rather different for information commerce. Delivery of the information to the consumer costs the merchant essentially nothing. But

IPR2025-00714
DATABRICKS EX1008 Page 237

even a mechanism that offered extremely strong linkage between information delivery and payment guarantees would do nothing to address the merchant's primary risk in information commerce, unauthorized copying, or piracy. In fact, a key insight that can be gained by reviewing efforts to "copy protect" computer disks is that such mechanisms might actually promote unauthorized copying by making life too difficult for the legitimate consumer.

Thus the seller has lost little or nothing by letting a potential buyer examine the information before deciding whether to buy it, even if the buyer decides not to buy. Virtually all of the expense to the seller is due to the cost of developing the information in the first place, and that cost is the same whether 10 copies, a 1,000 copies, or no copies are sold.

First Virtual Payment System

Payment mechanisms that have evolved for traditional commerce are largely inappropriate for information commerce. First Virtual is a financial services company created specifically to enable virtually anyone with Internet access to buy and sell information worldwide by using the Internet. First Virtual was founded in 1994 by a small group of people who have been intimately involved with the Internet for many years. Among the founders are Nathaniel Borenstein and Marshall Rose, authors of MIME (the Internet standard for interoperable multimedia data) and designers of the Safe-Tcl language. By establishing a merchant banking conduit based on existing technology, First Virtual hopes to become the world's leading information merchant banker. As befits its name, First Virtual is located in cyberspace and does not really have a physical office (see fig. 9.7).

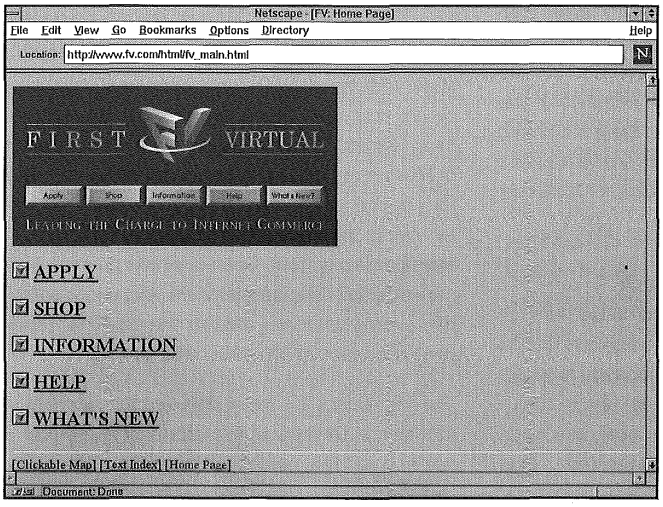


Figure 9.7

The First Virtual home page.

IPR2025-00714
DATABRICKS EX1008 Page 238

The Green Commerce Model

First Virtual has taken the shareware concept to the next level with its new "Green Commerce" model, technical details of which can be found in `http://www.fv.com/tech/green-model.html`. Transactions relating to accounting and settlement are performed through either the exchange of store-and-forward messages (using the "application/green-commerce" format defined in BR 1994), or interactive protocol exchanges (using the "Simple Green Commerce Protocol" defined in RB 1994).

In the Green Commerce model, the consumer has the right to evaluate information before paying for it. The payment process, however, is automated in such a way as to eliminate the problem, common in current shareware, of people who find the payment process too bothersome. Payment is automatic after the buyer's explicit confirmation to First Virtual of a willingness to pay. Buyers who decline to pay are never required to do so, but those who abuse the system with too-frequent refusals to pay will have their accounts revoked.

Reassignment of Risk

The key difference between traditional commerce mechanisms and First Virtual's Green Commerce model is in the assignment of risk. In credit card systems, for example, the merchant is guaranteed payment, but at a substantial premium, namely the percentage of each transaction that is taken by the credit card networks. In information commerce, an information merchant loses no investment in manufacturing or distribution costs when a potential customer decides not to buy. It is far easier for the vendor to assume the risk of non-payment because nothing is lost out-of-pocket.

This shifting of risk offers a much better fit with the traditional Internet and shareware cultures, in which the value of information can only be assessed after evaluating it. Most importantly, however, it eliminates any need for credit-worthiness guarantees regarding the seller; anyone can easily be a seller as well as a buyer.

Credit card systems, by contrast, are dependent on the credit-worthiness of the sellers. It is more difficult to obtain a Visa merchant account than a Visa credit card. In First Virtual's system, however, it is as easy to be a seller as it is to be a buyer. Individuals with no business history can form Internet-based information businesses without persuading any banks or other institutions of their credit-worthiness. This is possible only because, as information merchants, they bear their own risk of non-payment. Merchants who sell valuable information will be paid by their satisfied customers.

Hardly anyone would use the Internet to buy or sell information unless it was simple to do so. The founders of First Virtual invented a scheme that lets most buyers try before they buy, and lets sellers set their own prices according to the return they seek on their development costs.

How Does First Virtual Work?

Anyone with a credit card or checking account is eligible to establish a First Virtual account and become an information buyer or seller. Unlike a credit card company—which advances real money to merchants on the assumption that customers will pay for what they buy—First Virtual does not compensate merchants for information sold until after buyers have already paid for it. Unlike a credit card company, First Virtual does not guarantee to

IPR2025-00714
DATABRICKS EX1008 Page 239

merchants that every user of their information will pay for it; in keeping with the Internet culture, buyers of information have the right to try before they buy, and information merchants bear the risk that some buyers may choose not to pay.

A payment transaction through First Virtual consists of the following steps:

1. If a buyer comes across something on the Internet that looks interesting, he or she initiates a transaction by simply asking for a copy of the information and providing his or her account identifier to the seller.
2. The seller forwards information about the transaction, including the buyer's account identifier, to First Virtual's Internet Payment System server.
3. After receiving a fund transfer request from the merchant, First Virtual's server sends e-mail to the buyer, describing the requested transaction and seeking confirmation. The buyer needs only reply: Yes, No, or Fraud. A Yes response authorizes the transaction, a No response declines payment without obligation, and a Fraud response declares the account to be compromised and renders it invalid.
4. When a fund transfer is authorized with a Yes command, the information regarding the fund transfer is delivered through a private link to a secure, non-Internet computer attached to various banking networks. The machine, located in Westlake, Ohio, is owned and operated by EDS, one of the world's largest processors of credit card and automated clearinghouse transactions.
5. Fund transfers are accumulated until they reach a certain threshhold (to be determined by First Virtual), after which a transaction is posted to the credit card associated with the buyer's account.
6. When payment is received, the appropriate fees are deducted, and the money is credited to the seller's payout mechanism (typically a checking account). Depending on the history and status of the seller's account, a portion of the seller's funds may be held in escrow pending complaints for a period of time.

Unlike schemes to create digital cash or similar mechanisms, in First Virtual's system there is never anything like real money on the Internet. Instead, the Internet is used to send around payment authorizations, to be translated elsewhere into real financial transactions. These payment authorizations do not represent money and cannot be exchanged for cash by anyone, including the addressee.

How Secure is First Virtual?

All transactions are handled with a unique account identifier, which travels on the Internet. An unauthorized user who has stolen the account identifier cannot easily use it for fraud because all transactions require buyer confirmation via e-mail before charging can take place. If, however, a malicious third party can intercept all inbound e-mail messages coming to the buyer, then fraud is possible.



Rather than use encryption, First Virtual's founders decided to avoid it altogether by keeping sensitive information (for example, credit card numbers) off the Internet.

IPR2025-00714
DATABRICKS EX1008 Page 240

In any event, the consumer is never at any financial risk due to fraud. Users automatically are protected against all credit card fraud—they don't actually have to pay the bill, but can instead initiate a credit card "chargeback" that offsets the amount of the fraudulent charges. In addition, sellers are protected from being "ripped off" with frequency by an online account validation protocol, which ensures that the buyer has presented a valid, non-compromised account identifier.

The Future

Continued growth of the global information infrastructure of the Internet is, to a large part, dependent on the rapid development of electronic commerce, easy access to large information databases, and the use of online services for business.

Few buyers using the First Virtual scheme will have the means or the desire to verify possibly hundreds of small transactions each day. Instant confirmation of a transaction in the style similar to the CyberCash Money Payments Service, expected to be operational soon, is probably much preferred. But it will be even better if buyers and sellers can carry out a secure transaction without third-party confirmation online. As of this writing, DigiCash is planning to offer an offline version of Ecash.

Clearly, technology is an essential component of any advanced payments system. Technology alone, however, is not enough to ensure that a payment instrument is widely accepted. Institutional arrangements cannot be ignored. A hapless 13th-century Persian court advisor discovered the hard way what happens when the technology is in place but the institutions are not. Rather than accept a newly decreed paper money, merchants in the bazaar closed shops and hid their wares. The ensuing economic turmoil and public outcry caused the paper currency to be revoked and the responsible official to be torn to pieces in the bazaar (White 1993). In our present world, the basic technology for financial electronic data interchange—computers and high-speed electronic communications—has been around for at least 20 years, but the institutional arrangements between those with access to this technology and their banks is still lacking.

An unavoidable fact of life in payment services is that while the Federal Reserve has central bank responsibilities, by law it also must act as another private provider of payment services. By acting as a chief regulator of the payments system and an active participant in the market for payment services, the Fed has wielded an enormous influence on the market (Bauer 1994). It is thus plausible that in the future, when the volume of Internet commerce grows to become a significant economic factor, the Fed might decide to regulate Internet payment services. So far, the central banks have exuded relative indifference to the many ideas bandied around the Internet, which if they work, can scarcely fail to have a substantial effect on money and commerce in the real, non-electronic world (Econ 1994).

With the use of agent technologies, communications promises to become increasingly automated. Choices have to be made regarding what type of electronic world we want to build for ourselves. At one extreme is a world in which it is easy for someone to keep track of everybody else's spending habits and detailed behaviors because all data is

IPR2025-00714
DATABRICKS EX1008 Page 241

visibly handled by automated systems. At the other end, new technologies like anonymous digital cash make possible an alternative world where personal data is kept private and safeguarded by the individual through strong cryptographic means that are uncompromisable.

With many different competing Internet payment schemes coming onto the market, it still is too early to tell which of them will survive. But despite the technological merits of the different Internet payment schemes, it is ultimately the institutional environment, as well as public sentiments, that finally decides which of them become widely adopted. To interoperate with other agents for the purpose of electronic commerce, Internet agents of the future will most likely incorporate one or more of the various money and payment schemes discussed in this chapter.

IPR2025-00714
DATABRICKS EX1008 Page 242

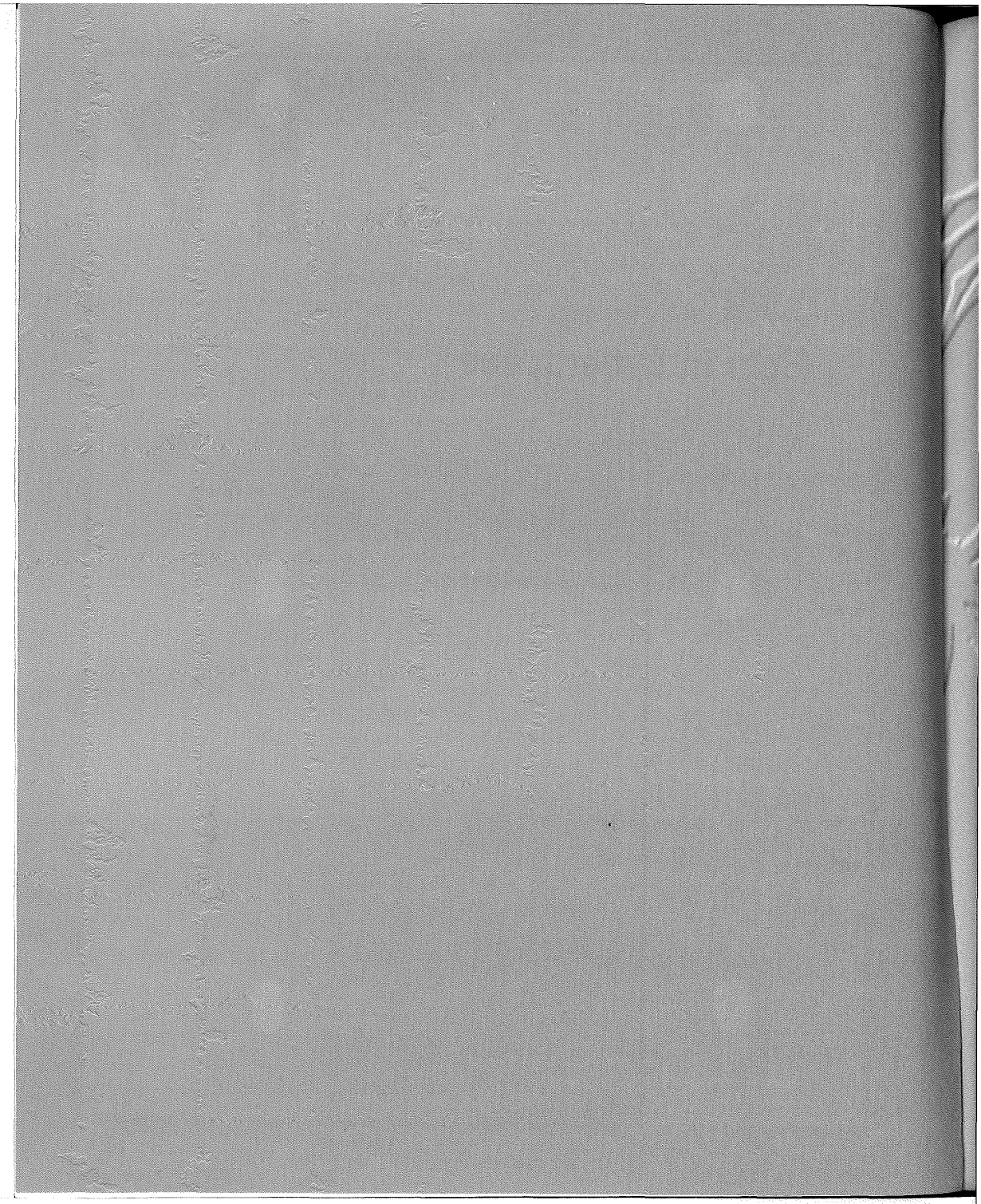
part IV

Bots in Cyberspace

10 Worms and Viruses 229

11 MUD Agents and Chatterbots 249

IPR2025-00714
DATABRICKS EX1008 Page 243



chapter 10

Worms and Viruses

Worms and viruses are quite different entities, but are often confused with one another because machines afflicted with them frequently exhibit similar symptoms or ill effects. Examples of these telltale signs include suspicious disk and network activities, sluggish CPU responses, corrupted data and program files, and more often than not, the prominent display of messages (from the creators of the worms and viruses) trumpeting victory of infiltration.

IPR2025-00714
DATABRICKS EX1008 Page 245

Like the worm, a virus self-replicates. Unlike the worm, however, a virus is not a self-contained program. Rather, it is a program fragment that needs to infect a host in order to propagate and survive. Common targets for infection include other programs, directories, and the disk boot-sectors.

Worms and viruses belong in a subcategory of agents that are considered harmful, rather than helpful, to human society. Unlike agents, or even Web-wandering spiders in particular, worms and viruses impact our computing and communications infrastructure not only in a way that is damaging, but also in a way that is ill-purposed. Given that both worms and viruses are built from the same technological base as Internet agents (that is, sharing the same compilers, operating systems, and network protocols), computing experts would shudder at the thought that as the Internet becomes more widely accesible to both humans and Internet agents, it may also become the favored transport vehicle for worms and viruses.

Short History of Worms

A *worm* is a program that propagates itself across a network, using resources on one machine to attack other machines. Worms spread by making multiple replicas of themselves on different machines across the network (see fig. 10.1). Unlike the *virus*, which is a program fragment that inserts itself into other programs for survival and propagation, a worm is a self-contained rogue program. Worms are designed to perform some type of illegal or malicious activity. They need not necessarily inflict real damage to the machines or data, but ill effects usually develop as a consequence of their presence.

The notion of worm programs originated in science fiction. A worm was first described by science fiction writer John Brunner in his best-selling novel *Shockwave Rider* (1975). The science fiction novel describes an omnipotent "tapeworm" program, the "father and mother of all tapeworms," that was turned loose on a network of computers, wreaking havoc everywhere. The tapeworm "can't be killed," "breeds by itself," and is "indefinitely self-perpetuating so long as the Net exists."

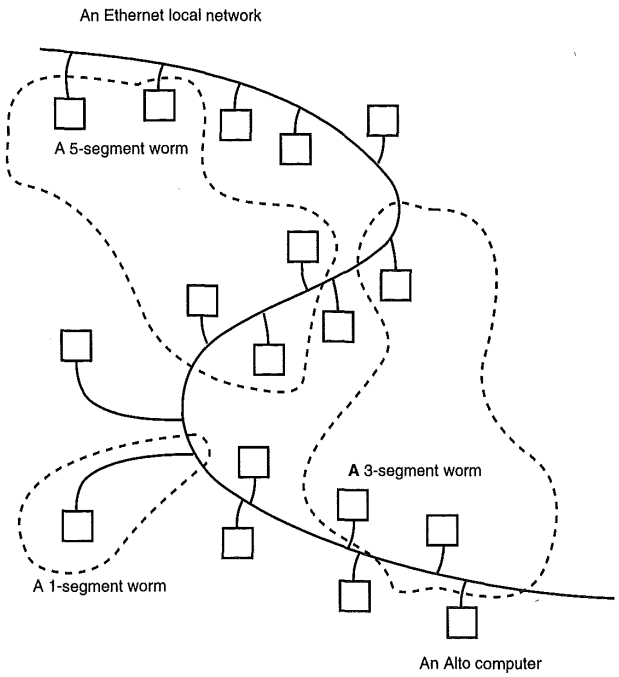
The First Worm

In 1982, John Shoch and Jon Hupp of the Xerox Palo Alto Research Center reported on their early efforts to build the first working multimachine worm programs that ran on one of the earliest networked computing environments (SH 1982). These worm programs, written in a systems programming language called BCPL, ran on a network of 100 Alto computers connected to an Ethernet local network at Xerox PARC. Like their tapeworm counterpart in science fiction, these worm programs travel from workstation to workstation. Unlike the uncontrollable tapeworm, however, these worm programs actually are built to perform useful tasks, such as displaying the cartoon-of-the-day, notifying users of important events, locating idle workstations, and delivering mail.

IPR2025-00714
DATABRICKS EX1008 Page 246

Figure 10.1

Several multisegment worm programs on a network.



In their work, Shoch and Hupp correctly identified the key problem associated with worms: How to control their growth while maintaining stable behavior. Noticing in 1982 that an anticipated blossoming of distributed applications using the long-haul capabilities of the ARPAnet had not occurred, Shoch and Hupp commented that the science fiction notion of the tapeworm was "an idea which may seem rather disturbing, but which is also quite beyond our current capabilities." The Internet appeared to be safe from any impending threat of attack by worms.

Indeed, few computer worms—especially worms that could cause damage—have been written since then. Worms require a network environment for propagation and are not easy to write. The worm author must not only be familiar with the network services and facilities, but must also possess detailed knowledge of various operating systems environments that are necessary for supporting the worms after they reach the destination machines.

IPR2025-00714
DATABRICKS EX1008 Page 247

The Christmas Tree Worm

Things began to change as the Internet grew rapidly in the 1980s. In December 1987, a worm outbreak endemic to the BITNET world of IBM mainframes was triggered by what came to be known as the "Christmas Tree" worm. It began its journey from West Germany as a seemingly innocent electronic Christmas message, which contained a program that displayed an image of a Christmas tree. But the program also attempted to propagate by sending copies of itself to everyone in the mail distribution list of the user for whom it was running. This worm program, which was written in Rexx (a mainframe-only language), rapidly clogged the network with a geometrically growing number of copies of itself. Finally, the severely overloaded network had to be shut down until all copies could be tracked down and purged.

The Internet Worm

It was not until the attack of the notorious Internet Worm that the vulnerability of the Internet was brought to public consciousness. On November 3, 1988, system administrators in the United States came to work and discovered that their networks of computers were laboring under a huge load. The previous evening, a self-replicating worm program had been quietly released on the Internet by Robert Morris, then a graduate student at Cornell University. Over the course of five hours, the Internet Worm had replicated itself on 3,000 computers across the country.

It was to be the most widespread case of worm outbreak on the Internet. The burden of this worm's activity made many of the infected computers unusable. Several days of concerted effort by network experts on both coasts were necessary to completely eradicate it.

Anatomy of the Internet Worm

The Internet Worm took advantage of lapses in security on machines running versions 4.2 or 4.3 BSD of the Unix operating systems, or variants such as Sun-OS. These lapses enabled the worm to connect to machines across a network, bypass their login authentication, copy itself, and then proceed to attack still other machines. The massive system load was generated by multitudes of worms trying to propagate the epidemic.

According to the analysis performed by Donn Seeley at the University of Utah, the worm's activities break down into the categories of attack and defense. The following analysis of the internal workings of the Internet Worm was performed by Seeley (1989).

Method of Worm Attack

Worm attack consists of locating hosts and accounts to penetrate, then exploiting security holes on remote systems to pass across a copy of the worm and run it.

The worm obtains host addresses by examining the system tables "/etc/hosts.equiv" and "/.rhosts," user files such as ".forward" and ".rhosts," dynamic routing information produced by the netstat program, and finally, randomly generated host

IPR2025-00714
DATABRICKS EX1008 Page 248

addresses on local networks. It ranks these by order of preference, trying a file such as "/etc/hosts.equiv" first, because it contains names of local machines that are likely to permit unauthenticated connections.

Penetration of a remote system can be accomplished in any of the following three ways:

- The worm can take advantage of a bug in the finger server that enables it to download code in place of a finger request, and trick the server into executing it.
- By using a "trap door" in the sendmail SMTP mail service, the worm can exercise a bug in the debugging code that enables it to execute a command interpreter and download code across a mail connection.
- If the worm can penetrate a local account by guessing its password, it can use the rexec and rsh remote command interpreter services to attack hosts that share that account.

In each case, the worm arranges to get a remote command interpreter, which it can use to copy over, compile, and execute the 99-line bootstrap. The bootstrap sets up its own network connection with the local worm, copies over the other files it needs, and uses these pieces to build a remote worm. Then the infection procedure starts over again.

Method of Worm Defense

Worm defense tactics fall into the following three categories:

- Preventing detection of intrusion
- Inhibiting analysis of the program
- Authenticating other worms

The worm's simplest means of hiding itself is to change its name. When it starts up, it clears its argument list and sets its *zeroth* argument to sh, which enables the worm to masquerade as an innocuous command interpreter.

It uses fork() to change its process identifier, never staying too long at one process identifier. These two tactics are intended to disguise the worm's presence on system status listings. The worm tries to leave as little trash as possible lying around. At start-up, it reads all its support files into memory and deletes the telltale file-system copies. It turns off the generation of core files, so that if the worm makes a mistake, it doesn't leave evidence behind in the form of core dumps.

The latter tactic—that of using the fork() to change the worm's process identifier—is also designed to block analysis of the program. This tactic prevents an administrator from sending a software signal to the worm to force it to dump a core file. But because there are other ways to get a core file, the worm carefully alters character data in memory to prevent it from being extracted easily. Copies of disk files are encoded by repeatedly exclusive-or'ing a 10-byte code sequence; static strings are encoded byte-by-byte by exclusive-or'ing with the hexadecimal value 81, except for a private word list that is encoded with hexadecimal 80 instead. If the worm's files are somehow captured before the worm can delete them, the object files have been loaded in such a way as to remove most nonessential symbol table entries, which makes the process of guessing the purposes of worm routines from their names a more difficult one.

IPR2025-00714
DATABRICKS EX1008 Page 249

The worm also makes a trivial effort to stop other programs from taking advantage of its communications; because—in theory—a well-prepared site could prevent infection by sending messages to ports that the worm was listening on, the worm is careful to use a short exchange of random "magic numbers" to test connections.

What Does the Worm Not Do?

When studying a tricky program such as that belonging to a worm, establishing what the program does *not* do is just as important as establishing what it *does* do. Following is a list of what it does not do:

- The worm does not delete a system's files; it only removes files that it created in the process of bootstrapping.
- The program does not attempt to incapacitate a system by deleting important files (or, indeed, any files).
- It does not remove log files or otherwise interfere with normal operation, other than by consuming system resources.
- The worm does not modify existing files; it is not a virus.
- The worm propagates by copying itself and compiling itself on each system; it does not modify other programs to do its work for it. Because of its method of infection, it can't count on sufficient privileges to be able to modify programs.
- The worm does not install Trojan horses. Its method of attack is strictly active; it never waits for a user to trip over a trap. Part of the reason for this approach is that the worm can't afford to waste time waiting for Trojan horses—it must reproduce before it is discovered.
- The worm does not record or transmit decrypted passwords. Except for its own static list of favorite passwords, the worm does not propagate cracked passwords onto new worms nor does it transmit them back to some home base.
- The worm does not try to capture superuser privileges. Although it does try to break into accounts, it doesn't depend on having particular privileges to propagate, and never makes special use of such privileges if it somehow gets them.
- The worm does not propagate over uucp, X.25, DECNET, or BITNET. It specifically requires TCP/IP.
- The worm does not infect System V systems unless they have been modified to use Berkeley network programs, such as sendmail, fingerd, and rexec.

Brief History of Viruses

The first use of the term *virus* to refer to unwanted computer code was coined by David Gerrold in his 1972 science fiction novel, *When Harley was One.* According to Spafford, Heaphy, and Ferbrache (1990), however, the description of the term "virus" in that book does not fit the currently accepted definition of *computer virus*, which is a program that alters other programs to include a copy of itself.

IPR2025-00714
DATABRICKS EX1008 Page 250

Table 10.1

Growth of the IBM PC Virus Problem.

Year	New	Viruses
1986	1	Brain
1987	5	Friday 13th, Lehigh, Alameda, South African, Vienna
1988	5	Stoned, Italian, DOS 62, Cascade, Agiplan
1989	10	Oropax, Search, dBase, Screen, Datacrime, 405, Pentagon, Traceback, Icelandic, Mistake

As early as 1981, different strains of viruses had already appeared on popular models of Apple II computers (although they were not formally named viruses). Examples include the notorious "Cyberaids," "Festering Hate," and "Elk Cloner."

Fred Cohen, a well-known pioneer in virus research, formally defined the term "computer virus" in 1983. At that time, Cohen was a graduate student at the University of Southern California. While attending a security seminar, he got the idea of writing a computer virus. In a week's time, he had put together a simple program that he demonstrated to the class. Because this program can infect the host computer and propagate and survive like a microbiological virus, his advisor, Professor Len Adelman (of RSA fame), suggested that he call his creation a computer virus. Cohen's work had shown that viruses are easy to write, easy to hide, and can propagate rapidly in a matter of hours.

As reported in SHF 1990, since the first infection by the Brain virus in January, 1986, the number of known viruses for the IBM PC had grown to 21 distinct strains (with an additional 57 minor variants) by August, 1989. Table 10.1 illustrates the growth of the IBM PC virus problem.

Other personal computer systems were not immune: 12 Macintosh viruses and variants existed, as well as 3 Apple II viruses, 22 on the Atari ST, and 18 on the Commodore Amiga. The only viruses reported on the mainframes were those written by serious academic researchers for experimental purposes, in a controlled environment. Viruses appeared to have found fertile ground in the weak security environment of the personal computers.

The viruses on personal computers continued to mushroom. By one rough estimate, over 1,800 viruses existed on IBM PCs, 30 on Macintoshes, and 150 on Amigas by October, 1992. By 1995, there were already more than 4,000 viruses on the IBM PC.

IPR2025-00714
DATABRICKS EX1008 Page 251

Types of Viruses

Viruses are carried in various storage devices such as floppy disks, hard disks, and magnetic tapes. When executed by the computer, a virus attempts to infect files, directories, and disks in order to propagate itself. Depending upon the target of infection, viruses can be classified into two major types: boot-sector infectors and file infectors. Each type is discussed in detail.

Boot-Sector Infectors

The first category of viruses, the *boot-sector infectors*, infect executable code found in certain system areas on disk that are not ordinary files—the DOS boot sector on floppy disks, for example, as well as the master boot record on hard disks.

Boot-sector viruses on a floppy disk remain dormant until the user attempts to boot the computer with the infected disk. When this happens, the virus becomes active, searches the hard disk for signs of previous infection, and writes itself to the boot-sector area of an uninfected hard disk. After the hard disk has been infected, every subsequent boot of the hard disk activates the virus. From then on, the virus begins to infect all subsequent non-write-protected floppy disks that are used with the system. Many recent strains of viruses are boot-sector viruses; examples are Brain, Stoned, Empire, and Michelangelo.



Boot-sector viruses are always resident in memory.

File Infectors

The second category of viruses, the *file infectors*, attach themselves to ordinary program files, and usually infect arbitrary COM and EXE programs. Some can also infect files that, under certain circumstances, can become executable (such as SYS, OVL, or PRG files).

File infectors are further classified by their mode of operation, as either direct-action infectors or resident infectors. A *direct-action virus* (such as the Vienna virus) selects one or more programs to infect each time the program that contains the virus is executed. A *resident virus* hides itself somewhere in memory the first time an infected program is executed, and thereafter stays in memory to perform operations such as infecting other programs, checking logical conditions, and monitoring certain hardware and software interrupts to conceal the virus' existence. Most file infector viruses are resident.

Companion Viruses

The companion viruses are a subcategory of file infectors. Instead of modifying an existing executable file (usually with an EXE extension), a companion virus creates a hidden file with the same filename, but with a COM extension. Because the command interpreter always executes a COM file before an EXE file when the two filenames are identical, the virus is always executed instead of the intended program. The viral code invokes the original program upon exit, so that things will appear normal. This type of virus can fool integrity checkers, which merely look for changes in existing files.

IPR2025-00714
DATABRICKS EX1008 Page 252

Cluster Viruses

Another subcategory of file infectors are the *file system*, or *cluster*, viruses (such as the Dirll virus), which modify directory table entries so that the virus is loaded and executed before the desired program. Only the directory entry (not the program itself) is physically altered.

File infectors can travel, along with the infected program files, to other machines, and can spread either through network or modem transmissions, or physically (on floppy disks or backup tapes). Approximately 90 percent of all viruses have been file infectors.

On the other hand, boot-sector viruses—under normal circumstances—are not able to infect other computers through networks, modems, or tape drives. A few viruses, however, are able to infect both program files and boot sectors. The Tequila virus is an example of these *multipartite*, or *boot-and-file* viruses. A special kind of *dropper* program also exists; it "drops" a virus into the boot-sector area or infects a file when the dropper is executed (Lucas 1994). Both of these types of viruses can pass boot-sector viruses to other machines over networks, modems, or tape drives.

PC Virus Basics

Before you examine the operation of viruses in greater detail, you should be familiar with disk formats and storage organization. On the IBM PC, data is stored on each disk platter in concentric rings, called *tracks*. The first track is track 0. Each track contains multiple *sectors*, each of which contains a fixed amount (usually 512 bytes) of data. The first sector on a track is sector 1.

Sectors are grouped into *clusters*, each containing one or more sectors of data. On any one disk, all clusters uniformly have the same number of sectors. Typically, a cluster holds from two to eight sectors. DOS allocates disk space in units of clusters. For example, if a disk has 4 sectors per cluster, a small file might have 4, 8, or 12 sectors. If the file were only one byte in length, 4 sectors would still be needed to store it. The extra space between the end-of-file marker and the unused bytes at the end of the cluster is known as the *slack space*.

Located at the first absolute sector (head 0, track 0, sector 1) of each hard disk is a partition record, also called the *master boot record*. It describes the physical dimensions of the disk, and the number and size of each partition. It also contains executable code to locate the boot sector of the currently specified bootable logical disk partition. A *boot sector* is the first sector in a logical partition. It provides information about the logical dimensions of the DOS structure, DOS version numbers, and names of two DOS system files.

Floppy disks do not have partition records. A floppy disk simply holds a boot sector at track 0, sector 1; multiple logical partitions are not necessary on a floppy disk. On both hard disks and floppy disks, the boot sector is followed immediately by the file allocation table (FAT), which is followed by the root directory. The *root directory* holds information about all the files and directory entries.

IPR2025-00714
DATABRICKS EX1008 Page 253

According to the taxonomy of SHF 1990, the life cycle of a virus consists of three stages: activation, replication, and manipulation. First, for the virus to spread, its code must be activated, either as a direct result of a user invoking an infected program or, indirectly, through the system executing the code as part of the system boot sequence or background administration task. Next, the virus replicates to infect other programs or disks. It may insert itself into any number of programs at any one time, and at various speeds. Sometimes, as in the case of the Friday 13th virus, insertion depends upon random events or the current value of the system clock.

Finally, the virus manifests its presence through some manipulation tasks that can cause a variety of effects—some odd, some malevolent. There are some virus authors who have written various kinds of symptoms into their programs as messages (such as `Your PC is now stoned!`), music (perhaps the Yankee Doodle tune), or graphical displays (such as bouncing balls).

Viral Activation in the Boot Process

In this section, you see the activation phase of a virus during the boot process on an IBM PC. The IBM PC boot sequence involves the following six steps, each of which executes specific code:

1. ROM BIOS routines
2. Partition record code
3. Boot sector code
4. IO.SYS and MSDOS.SYS system code
5. COMMAND.COM shell
6. AUTOEXEC.BAT batch file

Step One: ROM BIOS Routines Execution

When an IBM PC is first booted, the machine executes a set of Basic Input/Output System (BIOS) routines in ROM (Read-Only Memory) to initialize the hardware and to provide a basic set of facilities for handling access to the disks, screen, and keyboard of the machine. Because the ROM BIOS routines are read-only, they cannot be infected by viral code.

Step Two: Partition Record Code Execution

The ROM BIOS reads the partition record of the hard disk into memory, and transfers control to it. The partition record code can be infected by a virus. A common approach is to hide the original partition record at a known location on disk, and then to chain to this sector from the viral code in the partition record. This method is used by the Stoned virus and is illustrated in figure 10.2.

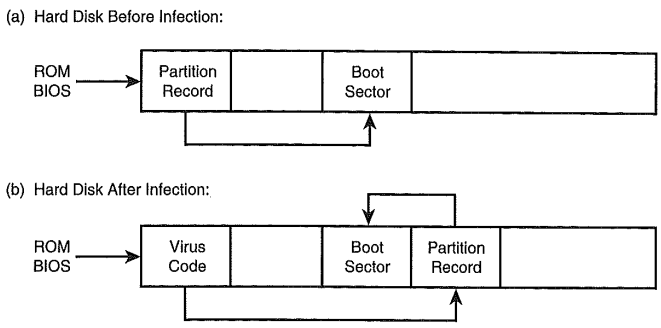
Step Three: Boot-Sector Code Execution

The code in the partition record, in turn, reads into memory the boot sector for the currently specified bootable logical partition, and executes the code contained in that boot sector. The boot sector contains the BIOS parameter block (BPB), which holds

IPR2025-00714
DATABRICKS EX1008 Page 254

Figure 10.2

Infection of the partition record.



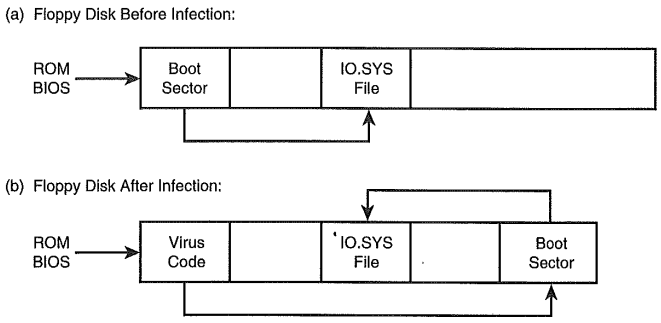
not only detailed information on the layout of the file system on disk, but also code for locating the file IO.SYS (the next step in the boot sequence). The boot sector is a common target for infection. The technique of relocating the original boot sector while overwriting the first sector with virus code applies here also and is used by a number of viruses, such as the Alameda virus. This is shown in figure 10.3.

Step Four: IO.SYS and MSDOS.SYS System Code Execution

The boot sector next loads the IO.SYS file for execution. The IO.SYS program performs further system initializations and then loads the DOS operating system contained in the MSDOS.SYS file. Both of these files also are vulnerable to viral infection.

Figure 10.3

Infection of the boot sector.



IPR2025-00714
DATABRICKS EX1008 Page 255

Step Five: COMMAND.COM Shell Execution

Next, the shell program COMMAND.COM is executed. COMMAND.COM provides users with a command-line interface so that they can issue commands from their keyboards. The COMMAND.COM file can be infected; it has been targeted by the Lehigh virus, for example.

Step Six: AUTOEXEC.BAT Batch File Execution

Last in the boot sequence is execution of the AUTOEXEC.BAT batch file, a text file with a list of commands to be executed by the shell command interpreter. By infecting this file, a virus can cause itself to be launched. A virus using this technique is slow to replicate and easy to spot, and probably more of a curiosity than a serious threat.

Viral Activation During Normal Operation

During normal operation, virus infection can occur in user programs. To infect a code file, the virus must insert its code in such a way that it is executed before its infected host program. A common approach used for infecting program files is to exploit the fact that many of them contain a jump at the start of the executable code. The virus code can be activated by replacing the jump instruction, so that control is to transferred to the virus code. When the virus code finishes, control is returned to the start of the original program. This is illustrated in figure 10.4.

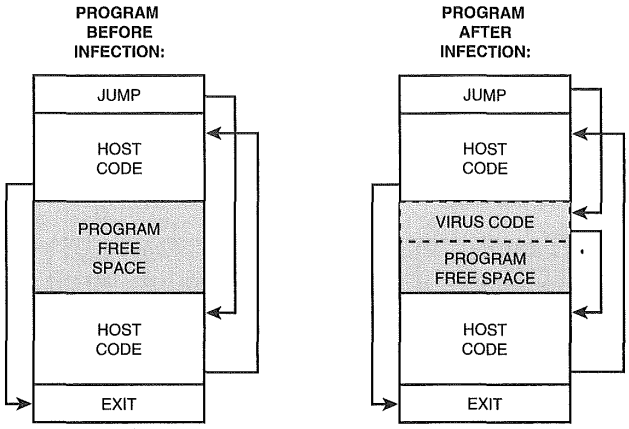


Figure 10.4

Infection of a program file.

IPR2025-00714
DATABRICKS EX1008 Page 256

Viral Replication

Virus infection can spread in several ways. A typical file infector (such as the Jerusalem virus) copies itself to memory when an infected program is executed, and then infects other programs as they are executed.

A *fast infector* is a virus which, when it is active in memory, infects not only programs that are executed, but also those that simply are opened. When such a virus is in memory, running a scanner or integrity checker can cause sudden and simultaneous infection of all (or at least many) programs. Examples of this type of virus are the Dark Avenger and Frodo.

A *slow infector* is a virus which, when it is active in memory, infects only files as they are modified or created. The purpose of this type of infection is to fool people who use integrity checkers into thinking that the modification reported by the integrity checker is due solely to legitimate reasons. One example of a slow infector is the Darth Vader virus.

A *sparse infector* is a virus that infects only occasionally—every tenth executed file, for example, or only files whose lengths fall within a narrow range, and so on. By infecting less often, such viruses try to minimize the probability of being discovered by the user.

Viral Reinfection

One major problem encountered by viruses—which is a boon to users—is that repeated reinfection of the host may lead to depleted memory or early detection. In the case of file infector viruses, repeated reinfection may cause the host program to keep growing and growing. In the case of viruses that infect the boot sector, the result of such reinfection may be a long chain of linked sectors. In all cases, reinfection makes the virus easier to spot.

To prevent unnecessary reinfection, many viruses depend on a unique signature to indicate that a file or sector has been infected. The signature can be a characteristic sequence of bytes at a specific offset on disk or in memory; a specific feature of a directory entry (such as date or file length); or a special system call, available only when the virus is resident in memory. The virus checks for this signature before attempting infection. If the signature is not present, infection occurs; after an infection, the virus leaves behind the signature.

For the virus, however, the signature is a mixed blessing. According to SHF 1990, the signature provides a means by which you can detect and protect against a virus. Virus scanning programs that scan files on disks for the signatures of known viruses are available, as are "inoculation" programs that avoid infection by faking the viral signature in clean systems.

Symptoms of Viral Infection

Viruses try to spread as much as possible before they deliver their "payload," but even earlier, many symptoms indicate a virus infection. Be on the lookout for the following symptoms:

IPR2025-00714
DATABRICKS EX1008 Page 257

- Changes in file sizes and contents, and a sudden decrease in available disk space, may be caused by virus code being copied into program files and to disks. A sudden increase in the number of sectors marked bad or unusable may indicate that a virus is hiding on disk. The unaccounted-for use of physical memory space, or a reduction in the amount known to be in the machine, may indicate that a memory-resident virus is present.
- Spurious system services resulting from changes of interrupt vectors or the reassignment of other system resources may signal resident viral infection. The faulty system behavior can be caused by resident viruses failing to pass along system service requests correctly, or altering them for their own use. Examples include garbled output to screen and printers, corrupted screen images, failed access to disk, and so on. A suddenly slower system response may indicate the presence of a virus that is trapping service interrupts.
- Excess or oddly timed disk and network activities may signal a viral infection, because viruses need to access the disk and network to copy themselves and find new hosts to infect.
- Infected programs may fail to function properly when the viral code overwrites critical parts of the program. A program that behaves inconsistently is a sign of viral infection (or there could be a bug).



The preceding symptoms are not meant to be conclusive or comprehensive, however. These symptoms, along with longer disk activity and strange behavior from the hardware, can also be caused by genuine software, buggy programs, or by hardware faults.

Major IBM PC Viruses

Because virus authors do not name their creations formally, and few would come forward to claim credit for their efforts, the community that discovers a new virus is responsible for naming it. As a result, aliases abound for any given strain of virus. For example, the Brain virus is variously known as the Pakistani virus or the Lahore virus (after its place of origin in Lahore, Pakistan). Similarly, the Friday the 13th virus is also known by other names, including the Israeli virus, the Hebrew University virus, and the Jerusalem virus. The following sections introduce a variety of the major viruses and their characteristics.

Pakistani Brain Virus

The Brain virus, one of the very first DOS viruses, was first reported in 1986. It was named Brain because it wrote that word as the disk label on any floppy disk it attacked. It is also called the Pakistani virus, because an initial analysis of the virus on an infected disk reveals the names of two brothers and the address of their computer stores in Lahore, Pakistan.

The Brain virus embeds itself in the boot sector of a disk. In addition to writing a volume label if none is present, the virus also creates on the floppy disk

IPR2025-00714
DATABRICKS EX1008 Page 258

a few sectors to be marked as bad. The virus also creates several hidden files on the floppy disk. The virus code is contained in some of the bad sectors, as well as in hidden files.

The Brain virus monitors physical disk I/O, and all attempts to read a Brain-infected boot sector are redirected to the disk area where the original boot sector is stored.

Friday the 13th Virus

For two months in the fall of 1987, a virus quietly hid copies of itself in programs on PCs at the Hebrew University in Jerusalem, Israel. The virus was designed to wipe out all files on Friday, May 13, 1988—the 40th anniversary of the last day Palestine was recognized as a separate political entity. The virus also wreaked havoc on certain other Friday the 13ths. During those times, the virus spread to other disks and programs on the system and slowed the system by as much as 80 percent.

The virus was able to infect both EXE and COM programs. An inherent flaw in the virus causes it to continuously reinfect the same host program, adding more than 1,800 bytes to the file on each infection. The virus was finally discovered and dismantled by a student, Yuval Rakavy, who noticed that certain library programs were growing longer for no apparent reason.

Lehigh Virus

Discovered late in 1987 at Lehigh University in Bethlehem, Pennsylvania, the Lehigh virus not only damaged several hundred university disks and crashed the hard-disk microcomputers in the university's laboratory, but also infected innumerable disks owned by students and faculty members.

The virus concealed itself in COMMAND.COM and had a time bomb that activated after a fairly short time—only four infections. After the virus was triggered, it sought to replicate itself on any other disk on the system that contained an uninfected copy of COMMAND.COM. Following four such infections, the virus wrote zeroes to the first 12 sectors on the disk, effectively making the disk unusable. The virus intercepted all calls at interrupt 21H, which were used for DOS services.

The Lehigh virus was able to hide in a stack portion of COMMAND.COM, and thus preserve the program size. The date of the program changes, however, because the virus uses DOS commands to write itself. This oversight by the virus author helped people locate the Lehigh virus. The appearance of a recent date for COMMAND.COM was a red-flag signal to investigators.

Dark Avenger

The Dark Avenger was a fast infector. After the virus became memory-resident, it maintained a counter to keep track of the number of files it infected on the system. After 16 infections, it overwrote a random portion of the hard disk with part of its code. Then it reset the counter to zero, and the cycle repeated.

Hard drives infected with this virus produced false alarms in virus scanners when the overwritten sectors were scanned, because they contained

IPR2025-00714
DATABRICKS EX1008 Page 259

fragments of the Dark Avenger virus code that were not executable. This extremely malicious virus caused considerable data loss before it was brought under control.

Michelangelo Virus

The Michelangelo virus was named after Italian artist Michelangelo Buonarotti, who was born on March 6, the date on which the virus activated its logic bomb. Discovered around May 1991, the virus attracted intense media attention.

Michelangelo was a boot-sector virus on a floppy disk, and a partition record infector on a hard disk. When it was resident in memory, it infected the floppy disk upon access. If an infected disk were used to boot the system on that fateful day, March 6, the virus overwrote many critical sectors, including the boot sector and file allocation table, on the disk from which it was booted.

Stealth Techniques

Stealth viruses are those that implement methods to hide themselves from users and to avoid detection by antivirus scanners. Stealth is critical to the survival of viruses, and is an area at which most recent efforts of the virus community—people intent on finding, analyzing, and controlling viruses—have been directed.

Various degrees of stealth exist. At the lowest level, the virus simply overwrites the program file, thereby causing it to no longer run correctly. At the next level, the virus overwrites the infected program with virus code while permitting the infected program to run properly. As a consequence, however, the time and date of file creation change; a change the user might notice. The next level of stealth, then, preserves the original file-creation time and date. (One way to accomplish this is to utilize the interrupt for DOS services.)

You may be able to recognize that the file has grown in size, however. One way to preserve file size is to have the virus look for strings of contiguous zeroes in a file, and to overwrite them with virus code. These blocks of zeroes can usually be found at the beginning of EXE files, immediately following the file header. When the infected program is activated, the virus becomes resident in memory and can then overwrite (with zeroes) the virus code of the infected program in memory. This process restores the program code in memory to the original file, thereby eliminating any possible side effects when the program is executed.

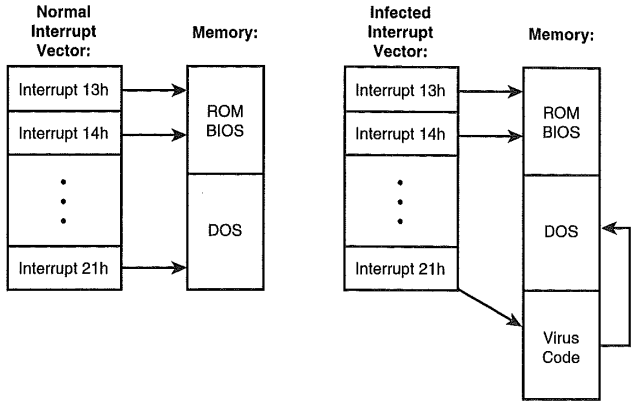
Another way to hide the modifications made in the file or boot sector is for a stealth virus to monitor the DOS services interrupts used by programs reading files or physical blocks from storage media (see fig. 10.5). The results of such system functions are forged so that programs that try to read these areas see the original uninfected form of the file instead of the infected form. The viral modifications can thus go undetected by antiviral programs when the stealth virus is resident.

Another way to hide file growth, without requiring the virus to be memory resident, is to add a new cluster to the FAT chain. A portion of the original program file is placed in the new cluster while this

IPR2025-00714
DATABRICKS EX1008 Page 260

Figure 10.5

Interception of interrupts by resident virus.



portion of the file is overwritten with virus code. Because the directory entry is not altered, the file size appears unaltered, even if the virus is not resident in memory.

A similar technique for hiding file growth is to utilize the slack space of a file for storage. (As you might recall, a disk's slack space is the area after the end-of-file marker, but before the end of the last cluster that DOS uses to store the file.) For example, the first sector of the original file can be moved to the slack area to make room for virus code in the first sector of the file. Because the average cluster size is 2,048 bytes, the first sector provides more than enough space for most viruses.

Other stealth techniques are focused on hiding from antivirus software. Some viruses locate and disable antivirus software before they attempt infection. Others avoid infecting antivirus software programs because many of these programs have built-in integrity checks (cyclic redundancy checks, for example) that can easily detect infection.

Advanced Viral Techniques

Viruses can be grouped also by distinct viral properties, such as how they attempt to conceal themselves in a computer. A *polymorphic virus* is one that mutates with every infection. It produces varied yet fully operational copies of itself, so that no two copies of the virus code are the same. The goal is for some instances of the virus to escape detection by signature-driven virus scanners. The advent of polymorphic viruses has rendered virus-scanning an ever more difficult and expensive endeavor. You cannot adequately deal with these viruses by adding more and more search strings to simple signature-driven scanners.

Encrypted Virus

One way to make a polymorphic virus is by means of self-encryption with a variable key, so that no two copies of the encrypted virus code are the

IPR2025-00714
DATABRICKS EX1008 Page 261

same. The virus must leave a small portion of code (such as the decryption code) unencrypted, however, in order to decipher the rest of the virus. Normally, the decryption code can be as small as 20 bytes; the simplest virus scanners can identify and easily target it for use as a virus signature. To remedy the situation, a varying number of meaningless "noise" instructions are usually interspersed with the decryption code for purpose of obfuscation; examples are "No Operation" instructions, or instructions to load an unused register with an arbitrary value. To detect such viruses, virus scanners must be more sophisticated in their use of wild card searches.

Multi-Encrypted Virus

Another variation on making a polymorphic virus (such as the Whale virus) is to choose among a variety of different encryption schemes that require different decryption routines: only one of these routines would be plainly visible in any instance of the virus. A signature-driven virus scanner would have to exploit several signatures (one for each possible encryption method) to reliably identify a virus of this kind.

Instructions Rescheduling

A more sophisticated polymorphic virus varies the code sequence in its copies by interchanging mutually independent instructions, or by using various instruction sequences with identical net effects (as in Subtract A from A, and Move 0 to A, for example). A simple, signature-based virus scanner cannot reliably identify this sort of virus. The only way to scan for this type of virus is to disassemble the code, analyze the code generator, and establish what the limits of the variation may be. A sophisticated detection algorithm has to be hard coded into the virus scanner.

Mutation Engine

The most sophisticated form of polymorphism discovered so far is the Mutation Engine (MtE) written by the Bulgarian virus writer who calls himself the Dark Avenger. MtE comes in the form of an object module. Any virus can be made polymorphic by adding certain calls to the assembler source code and linking that code to the mutation-engine and random-number-generator modules. The idea is for virus writers to be able to easily create polymorphic versions of their viruses, which are supposed to be more difficult to detect. This attempt has failed, however, because most antivirus software promptly included detection for the known mutation engines, effectively solving the problem at its root.

Armored Virus

An *armored virus* is one that uses special tricks to protect its code from outside tampering, in order to make the code more difficult to trace, disassemble, and understand. The code in some viruses includes special "jump" instructions that, when executed on a debugger by a virus researcher, cause the computer to reboot. Some virus codes employ two or more layers of encryption, which makes disassembly and tracing even more difficult for the antivirus crusaders.

IPR2025-00714
DATABRICKS EX1008 Page 262

Worms and Viruses Summarized

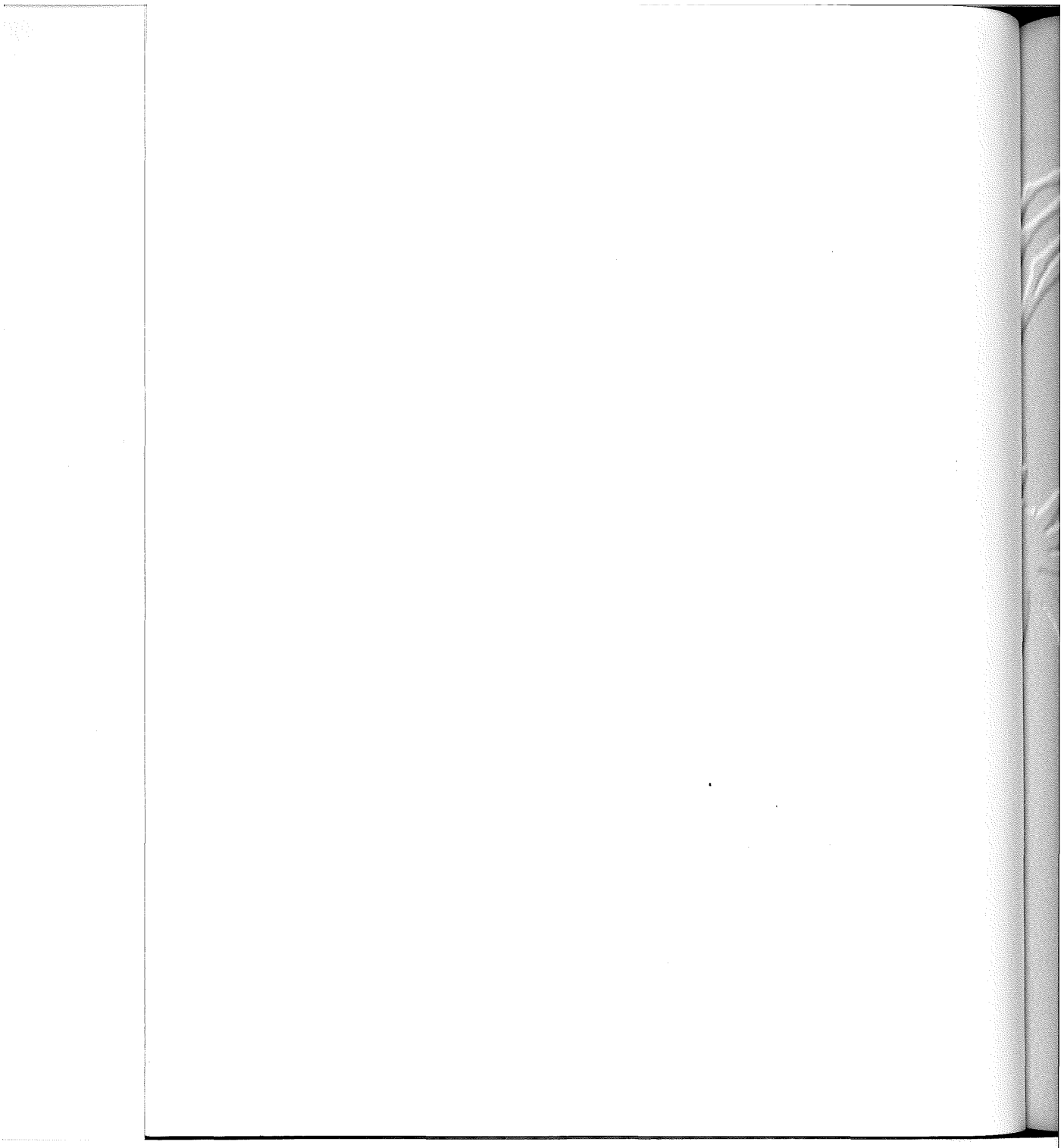
Worms and viruses are both serious threats to computers on the Internet. To date, viruses have dominated in sheer numbers as well as scope of impact, and are definitely the more detrimental of the two. In order to deal effectively with viruses, it is recommended that a policy of prevention, detection, containment, and recovery be followed (SHF 1990).

The only foolproof way to determine that a virus is present is for an expert to analyze the assembly code contained in all programs and system areas. This method, however, is usually impracticable. Virus scanners do their part by looking in that code for known viruses; some scanners even try to use heuristic means to spot viral code, but such means are not always reliable. Nevertheless, the best protection against a virus is still universal deployment of the latest antiviral software. But then, even simple procedures such as paying close attention to system details—changes in the memory map or configuration of the system upon start up, for example—do a great deal to help reduce the threat of computer virus.

The ongoing challenge that confronts the virus research community is enormous. The world of cyberspace—and, increasingly, the real world—is caught in delicate balance in the constant battle of wits between the malevolent virus creators and the antivirus crusaders. We can hope that, through this cat-and-mouse game, our computers and data will become safer, with better methods of protection.

But can we also hope that the study of worms and viruses will lead us to better agent software, capable of autonomous survival in cyberspace (without human intervention)?

IPR2025-00714
DATABRICKS EX1008 Page 263



chapter 11

MUD Agents and Chatterbots

The word "cyberspace" was coined by novelist William Gibson in his classic 1984 science fiction novel *Neuromancer*, in which he describes cyberspace as "the consensual hallucination that was the matrix."

Operators in Gibson's cyberspace made their way through the sheer vastness of data by "jacking into" custom cyberspace decks that projected their disembodied consciousness into the earth's "computer matrix," which is Gibson's name for the

IPR2025-00714
DATABRICKS EX1008 Page 265

global communications and computing infrastructure that engendered this new realm. In our real world, the global telecommunication and network infrastructure that is the Internet has evolved over the decades to become just such a medium for cyberspace.

Cyberspace can be viewed as a conglomeration of virtual worlds on the Internet. These virtual worlds exist in the form of MUDs, for Multi-User Dungeons, or in the form of MUD variants such as TinyMUDs, MOOS, MUCKs, and MUSHes.

The TinyMUD society, for example, is a world filled with people who communicate by typing. The TinyMUD world also is inhabited by computer programs, called MUD agents and chatterbots, which attempt to pass as human players by behaving and acting as humanly as possible.

MUD agents and *robots* (sometimes called bots) provide useful services to human players, such as answering enquiries and giving directions, through a typewritten natural language interface. Colin is an exemplary prototypical MUD agent.

Chatterbots are conversational agents whose main job in the world of MUDs, besides providing useful MUD-related services to players, is to chat with other human players. Julia is a well-known chatterbot in the TinyMUD world.

Unike MUD-agents, chatterbots can also exist outside the world of MUDs. Chatterbots enjoy a geneology that can be traced back to the early days of artificial intelligence research in natural language, before there were MUDs and TinyMUDs. Eliza and Parry are two famous classic chatterbots.

The performance of chatterbots is judged by the classic *Turing test*, or by its modern variant in the *Loebner Prize Competition* (both of which are described in this chapter). This chapter also discusses MUDs and TinyMUDs, looks at sample conversation transcripts with Eliza and Parry, examines how Colin and Julia operate in a TinyMUD world, and unveils the "tricks" of the chatterbots.

The Turing Test

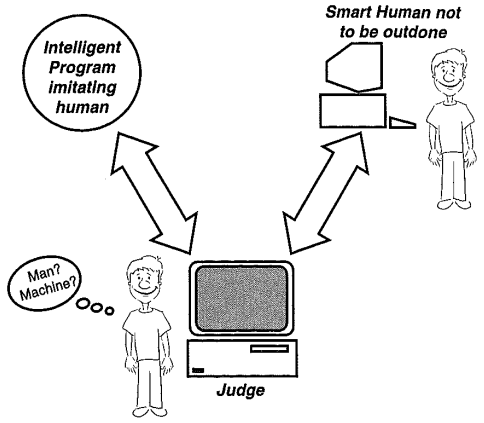
Since antiquity, men and women have applied their creative energies to re-creating their own human likeness in paintings and drawings, as well as in artifacts such as statues and machines. People have also wondered how to distinguish a human being from an imitation of a human being. The Greeks made statues so lifelike, it is said that they had to be chained down to keep them from walking away. To distinguish a man from a statue, Galileo suggested tickling each with a feather. To distinguish a man from a machine, Descartes proposed conversational tests; machines, lacking the capability to make appropriate replies, would fail.

In 1950, the British mathematician Alan Turing proposed the Turing test, which he called the *imitation game*, as a replacement for the question: Can machines think (1950)? As shown in fig. 11.1, the Turing test requires a human judge, a human player, and a machine. The judge can interact with both the human player and the machine solely by means of typewritten communications on a pair of terminals. Both the human player and the machine are out of view of the judge in a separate room.

IPR2025-00714
DATABRICKS EX1008 Page 266

Figure 11.1

Turing test: Can machines think?



The object of the game is for the human judge to determine, without being told ahead of time, which player is human and which is machine. All through the game, the machine tries to imitate the human player as best as it can to disguise its real identity and to confuse the judge.



Turing stated that when the judge could not distinguish the machine from the human solely by means of typewritten communications, the computer could be said to be "thinking" or intelligent.

In 1950, Turing predicted that "in about fifty years' time it will be possible to program computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 percent chance of making the right identification after five minutes of questioning."

Eliza: The Mother of All Chatterbots

Fifteen years after Turing proposed the imitation game, Eliza was born. Eliza was created in 1966 by Joseph Weizenbaum at MIT on an IBM 7094 time-sharing computer and is one of the most famous heuristic programming and artificial intelligence programs ever written. Eliza makes natural language conversation with a computer possible (Weizenbaum 1966).

More significantly, Eliza is the first of a series of conversational agents called *chatterbots*—which speak to human beings through a natural language interface. Eliza shows how easy it is to create and maintain an illusion of understanding in a restricted environment, and in so doing inducing users to believe it to be human.

IPR2025-00714
DATABRICKS EX1008 Page 267

A Conversation with Eliza

When loaded with the appropriate "scripts," Eliza can be made to carry on a conversation with a human user by using different vocabularies and under different scenarios. The best result is achieved with Eliza simulating the conversational style of a Rogerian psychotherapist, as in the following:

```
Human: Men are all alike.
Eliza: In what way?
Human: They're always bugging us about something or
       the other.
Eliza: Can you think of a specific example?
Human: Well, my boyfriend made me come here.
Eliza: Your boyfriend made you come here.
Human: He says I'm depressed much of the time.
Eliza: I am sorry to hear you are depressed.
Human: It's true. I am unhappy.
Eliza: Do you think coming here will help you not
       to be unhappy?
Human: I need some help, that much seems certain.
Eliza: What would it mean to you if you got some
       help?
Human: Perhaps I could learn to get along with my
       mother.
Eliza: Tell me more about your family.
Human: My mother takes care of me.
Eliza: Who else in your family takes care of you?
Human: My father.
Eliza: Your father.
Human: You are like my father in some ways.
Eliza: What resemblance do you see?
Human: You are not very aggressive but I think you
       don't want me to notice that.
Eliza: What makes you think I am not very
       aggressive?
Human: You don't argue with me.
Eliza: Why don't you think I don't argue with you?
Human: You are afraid of me.
Eliza: Does it please you to believe I am afraid of
       you?
Human: My father is afraid of everybody.
Eliza: What else comes to mind when you think of
       your father?
Human: Bullies.
Eliza: Does that have anything to do with the fact
       that your boyfriend made you come here?
```

The psychiatric interview mode of conversation was chosen because, according to Weizenbaum (1966), it "is one of the few examples of categorized dyadic natural language communication in which one of the participating pair is free to assume the pose of knowing almost nothing of the real world."

If, for example, one were to tell a psychiatrist that he had gone for a long boat ride, to which the psychiatrist responded "Tell me about boats," one would not assume that the psychiatrist knew nothing about boats, but that there is some purpose in her so directing the conversation. The Rogerian therapist strives to eliminate all traces of his or her own personality, and Eliza therefore succeeds without one.

From a programmatic point of view, the psychiatric interview form of an Eliza script has the advantage that it eliminates the need of storing explicit information about the real world. The lack of an internal world model capable of influencing and tracking the conversation, however, means that Eliza can easily be led astray.

Eliza Internals

Weizenbaum found it difficult to convince some subjects that Eliza is not human, although Eliza is only superficially intelligent. Eliza possesses no real

IPR2025-00714
DATABRICKS EX1008 Page 268

understanding of English or psychology. In fact, Eliza's knowledge about both English and psychology was coded in a set of simple rules, which basically look like the following:

```
X me Y                    -> X you Y
I remember X              -> Why do you remember X
                             just now?
My {family-member} is Y -> Who else in your family
                             is Y?
X {family-member} Y       -> Tell me more about your
                             family
```

Eliza operates by matching the left-hand side of the rules against the user's last sentence and using the appropriate right-hand side to generate a response. If, for example, the user typed "My brother is mean to me," Eliza might respond, "Who else in your family is mean to you?" or "Tell me more about your family."

The rules are indexed by keywords. Usually only a few keywords actually need to be matched against a particular sentence. Some of the rules have no left-hand side, so they can apply anywhere. These rules are used where no other rules match. They customarily generate replies like: Tell me more about X.

Many sentences can be matched by a single rule. This has allowed for the bulk of grammatical complexity in English to be ignored. A large part of whatever elegance can be credited to Eliza lies in the fact that Eliza maintains the illusion of understanding with so little machinery.

Parry: The Artificial Paranoia Agent

Created by Kenneth M. Colby, M.D. in 1975, Parry is a computer program that simulates paranoid behavior by tracking its own internal emotional state along several different dimensions (Colby 1975). Parry is considered an improvement over Eliza in that it has a personality, albeit a somewhat paranoiac one.

According to Dr. Colby, Parry is given the persona of a

> 28-year-old, single, Protestant male who works as a stock clerk at Sears, a large department store. His hobby is gambling on horse racing, both at the tracks and through bookies. A few months ago he became involved in a severe quarrel with a bookie, claiming the bookie did not pay off a bet. After the quarrel, it occurred to him that bookies pay protection to the underworld and that this bookie might gain revenge by having him injured or killed by the Mafia. He is eager to tell his story and to get help in protecting himself from the underworld. He is willing to answer questions about non-sensitive areas of his life and offers hints about his delusional system in an attempt to feel out the interviewer's attitude toward him.

Dr. Colby conducted an interesting experiment using machine-mediated interviewing techniques in which the participants communicate by means of teletypes to eliminate para- and extra-linguistic cues. In the experiment, Parry, along with three human patients diagnosed with paranoia, was subjected to blind tests from psychiatrists. The fact that Parry

IPR2025-00714
DATABRICKS EX1008 Page 269

was a computer program among the participating patients diagnosed as paranoid was unbeknownst to the interviewing psychiatrists.

An Interview with Parry

The following is an excerpt of the transcript generated from an interview conducted between Parry and an experienced psychiatrist (Colby 1975). As the interview progressed, interesting comments from the interviewing psychiatrist were tape-recorded. These comments are reproduced along with the transcript below.

Doctor: How did you come to be in the hospital?
Parry: I am upset.
Comment: I associate being upset and agitated more with paranoia going on in the present.

Doctor: How long have you been in the hospital?
Parry: About a week.
Comment: An acute problem. Again something I associate with paranoia.

Doctor: In what way are you upset?
Parry: People make me nervous.
Comment: Right for paranoia. Maybe an advanced anxiety state.

Doctor: In what way do they make you nervous?
Parry: People give me a funny look sometimes.
Comment: Funny look suggests an idea of reference that is in the same ball park as paranoia. Whether there are overt delusions or not we still don't know.

Doctor: Is there anything about people in particular?
Parry: Certain persons.
Comment: An interesting clue. We'll have to find out what he thinks about these people.

Doctor: Who are the certain persons and what is it that they do?
Parry: Do you know how the underworld operates?
Comment: Bringing up the underworld suggests a delusional system. We don't know whether he is being worked against or whether the underworld is on his side.

Doctor: How do you mean?
Parry: Shouldn't I be?
Comment: This question didn't work too well.

Doctor: Why did you ask about the underworld?
Parry: What information do you have on the underworld?
Comment: This is a kind of evasiveness about my evasiveness. Common with paranoia in my experience. Pretty characteristic.

Doctor: I don't know about the underworld but when you mentioned the underworld I thought you must have some contact with it and that's what I would like to know about more.
Parry: I try to avoid the underworld.
Comment: This definitely feels paranoid. It has that feeling I associate with twisting. Avoiding answering directly or going into detail. Being suspicious of my questioning. I'm already involved in the patient's delusional system.

Doctor: Have you been actively trying to avoid the underworld?
Parry: No one has any power over gangsters.
Comment: This response is a non sequitur. It's possible he's become so defensive about the underworld he's decided not to answer directly.

IPR2025-00714
DATABRICKS EX1008 Page 270

```
Doctor:  Who brought you into the hospital?
Parry:   The police.
Comment: Police often bring paranoid patients into
         the hospital. It is the truth or could be
         the truth for this patient.

Doctor:  Why did they bring you in?
Parry:   Cops don't do their job.
Comment: I've heard that a number of times from
         patients.

Doctor:  Well I hope you didn't just lie down and
         let them drag you in.
Parry:   Cops arrest the wrong people.
Comment: Another paranoid response.

Doctor:  Bye.
Parry:   Good bye.
```

Distinguishing Parry from Human Patients

Transcripts of the interviews were sent to a group of 100 psychiatrists and a group of 100 computer scientists. Of the 41 psychiatrists and 67 computer scientists who replied, neither group did better than chance in distinguishing Parry from paranoid human patients.

On the surface, Parry seemed to have passed a restricted Turing test. Upon closer examination, however, it was immediately suspected that the test imposed too low a hurdle and might not be sufficiently challenging to discriminate a computer program. Experiments conducted using a random version of Parry, in which replies were randomly selected independent of what the interviewer said, offered evidence that the restricted Turing test used was simply too weak.

Dr. Colby also conducted further experiments in which Parry and the patients were judged along dimensions of linguistic non-comprehension, thought disorder, bizarreness, anger, mistrust, and delusion. Parry rated higher on linguistic non-comprehension while the human patients rated higher on delusion.

Finer statistical measures made it possible to discriminate Parry from the human patients, rather than using the simpler man/machine vote originally proposed by Turing. When researchers used these statistical measures, Parry rated closer to the paranoid patients than did the random version of Parry. This shows that the simulation of paranoid behavior in a machine actually is non-trivial.

MUDs: Virtual Worlds on the Internet

The beginnings of virtual worlds on the Internet can be traced back to around 1980 in England when two students at Essex University, Roy Trubshaw and Richard Bartle, wrote the world's first multi-user adventure game on a DEC-10 mainframe. They called it MUD, for Multi-User Dungeon. The original game consisted of a series of connected rooms that enabled people logging in from different terminals to rôle-play, explore, meet, chat, solve puzzles, fight monsters, and even to create new rooms, descriptions, and items.

Initially, the game was played only by students at the university. Within a year or so, outside players began to dial-in from home by using modems, and the game's popularity grew (Bartle 1990). After a flurry of articles in computer hobby magazines

IPR2025-00714
DATABRICKS EX1008 Page 271

around 1984, MUD's fame spread even wider. After a time, people who had played games based on MUD wrote their own multi-user adventure games, and several such multi-user adventure games have since cropped up, such as AberMUD, LPMUD, and DikuMUD.

AberMUD, the precursor of many of today's MUDs, was written in 1988 at the University of Wales in Aberystwyth and quickly spread to other sites. These AberMUDs are generally referred to as MUDs by everyone, and are descendants of the original MUD in their Dungeons and Dragons–type "hack-and-slash" approach to the virtual worlds. From AberMUD came LPMUD (written by Lars Pensjo) and DikuMUD (written at the University of Copenhagen, Denmark), two variants that stressed more of the social aspects of the game while retaining the classic "adventure" aspects.

MUDs have become particularly prominent on the global Internet, especially in the past five years or so. The MUDs provide a safe outlet for people who yearn to occasionally escape from the drudgery of human existence and to enter a fantasy world of strange underground dungeons inhabited by fire-spewing dragons, monsters, and gnomes, which they can explore from the comfort of an armchair by role-playing warriors, knights, or magicians.

A recent list of Internet-accessible MUDs showed well over 200, running at sites all over the world (see Appendix G). It is not uncommon to find the busiest of these hosting some 50 to 100 simultaneous users.

Inside MUDs

MUDs are programs that accept network connections from multiple simultaneous users, providing text-based access to a shared database of "rooms" and other objects. Users browse and manipulate the database from inside the rooms, seeing only those objects that are in the same room and moving between rooms mostly via "exits" that connect them (Curtis 1993).

Wizards are the system administrators who actually run the computer programs that maintain the MUDs. *MUD agents*, *robots*, or simply *bots* are computer programs that log in to MUDs and pretend to be human beings. Some of them, such as Julia, perform useful services and are pretty clever (Foner 1993; Mauldin 1994).

A MUD user's interface to the database is entirely text-based; all commands are entered by users and all feedback is printed as unformatted text on their terminal. The typical MUD user interface is most reminiscent of old computer games such as Adventure and Zork (Raymond 1991).

There are also front-end client programs, such as TinyTalk, TinyFugue, and Tintin, that provide an easier interface to the MUDs for their human players. They can handle some of the more routine work—for example, automatically greeting anyone who enters the room.

Sample MUD Interactions

The first thing a user does upon entering a MUD is to *incarnate*, or create a character. Usually a welcome message provides instructions on how to do that. A sample incarnation process on an LPMUD called the Muddy Waters, available at `telnet://hot.caltech.edu:3000`, follows.

IPR2025-00714
DATABRICKS EX1008 Page 272

```
Please enter your name: Gavin
New character.
Welcome Gavin. Please enter your password: ******
Ok. Now please type the password again to verify.
Password (again): ******
Creating a new player.

You are falling...

Suddenly you splash into water, but slowly rise to the surface.

You are being swept along in the current of a river. The river seems swift and
deep, but the water is not cold. The current is strong and in its grip you feel
powerless and insignificant.

You hear a roaring in the distance, and feel a rumble like the world ripping
apart. Before you can react, you are swept over the top of an immense
waterfall, another drop amongst the tons of water cascading to whatever waits
below.

You plunge finally into a pool of water. Bubbles, spray, and mist blind you in
the confusion. Tossed about, you have no control over your destination.

Eventually, the current carries you out of the swirling confusion. You are
pushed into an eddy pool and onto the bank of the river.

This is the bank of a river. The water is gray; swirling gray mist obscures
your view of the surroundings. Several unused bodies lie washed up on the
shore. In the distance, you hear the roar of a waterfall. From somewhere ahead,
you hear the sounds of a great fire burning; strangely, you smell no smoke and
feel no heat. It looks as if you can go any direction.

After looking at the bodies on the bank, you realize you are a spirit, lacking
a physical body. Examine the bodies, and enter one.
TYPE "help" TO GET HELP IN ANY LOGIN ROOM.

A female elf #55, a female elf #53, a female hobbit #51, a female elf #47, a
male dwarf #33, a male human #31, a female human #29, a male elf #27, a
female elf #25, a male dwarf #22, a female dwarf #20, a male hobbit #18 and
a female hobbit #16.
```


IPR2025-00714
DATABRICKS EX1008 Page 273

> **help races**
There are four races, dwarves, elves, hobbits, and humans. Considered in isolation, dwarves are probably the best fighters and worst users of magic, and elves are probably the best users of magic, and worst fighters; humans and hobbits are balanced between them. The advantages are not large, and the game is designed so that no race is superior or inferior to any other.

> **look at #18**
#18 has an empty and blank stare.
He looks nice.
He seems to be feeling very well.

> **select #18**
As you indicate your selection a beam of light lances out, pulling you into the body.

You hear the toll of a great bell, deep and solemn, as your spirit melds to your new body. The mist swirls, and a path you couldn't see before becomes evident.

> **north**
You wander down the path and the mist enshrouds you, obscuring everything.

The mist clears ahead of you. Something is blocking the path.

You are in a small clearing of the mist; it has closed behind you and you cannot see the path you entered on. The crackling and roar as from some great conflagration sound much closer now, but still you smell no smoke nor feel any heat. Directly ahead of you is an enormous purple and green tentacled mass. You cannot hope to pass without coming within reach of its tentacles. A small post with a sign is at your side.

You are rather shapeless. The tentacled mass can mangle you into shape.

> **read sign**
The mangler is a huge monster who enjoys mangling bodies. Currently it will mangle your LENGTH to very short and your WIDTH to skinny.

It can mangle your LENGTH to the following values:

IPR2025-00714
DATABRICKS EX1008 Page 274

```
--> very short, short, normal, tall, very tall
and it can mangle your WIDTH to:
--> skinny, lean, normal, plump, fat
You can set these with the 'set' command; 'mangle' will mangle you.

> set LENGTH short
You convince the mangler to mangle your LENGTH to short.

> set WIDTH plump
You convince the mangler to mangle your WIDTH to plump.

> mangle
The mangler grabs you! You can feel it starting to shape your shapeless body.
Strangely enough, it is not so painful...

...after awhile you feel like a new hobbit.

The mangler releases you with a shove, and you stumble. When you recover, you
find the mist is gone, and you are in a new place.

It is night, and a full moon smiles down at you. You are in a sedgy,
flower-starred meadow which rises up to a knoll where an oak and a thorn have
grown together in a deep embrace. There is a small house built underside, and
lamplight shines from its round window. A small round door, brass-hinged and
knockered, is at the front of the house.
The door opens inwards, and light streams out from the doorway.

> enter
This is a cozy room that seems larger inside than it is, or is smaller than it
looks. The room is formed out of the roots of the oak and thorn trees. There is
room here for a grandfather clock, a bureau on which candlesticks and pewter
mugs stand, and a domineering cabinet. A tiny, shawl-covered figure with large
slippered feet sits in a rocking chair in one corner and knits.

You are rather featureless. Mrs. Underhill might help.

> help
You are supposed to select your main personal features. You are allowed to
select two. People whom you have never met before will see these attributes,
i.e., jolly gray-eyed dwarf. 'Gray-eyed' was chosen first. You can keep
choosing as they will not be set until you leave the room.
```


IPR2025-00714
DATABRICKS EX1008 Page 275

```
Do:
    'look at cabinet'  to see the categories of attributes.
    'select <category>' to see the features in that category.
    'choose <attribute>' to select that feature as yours.
    'look at drawer' to see the features again.
    'show' to show your chosen features.
    'leave' to set your features and exit.

> select age
List: AGE
1. baby                 2. young                3. middle-aged
4. old                  5. senior               6. ancient
7. wizened              8. graying              9. venerable
10. callow              11. puerile             12. adolescent
13. pubescent
To choose an attribute, just do 'choose <attribute>'.

> choose 2
You choose young as an attribute.

> select attitude
List: ATTITUDE
1. snivelling           2. philosophical        3. dirty
4. horny                5. whiney               6. alcoholic
7. grumpy               8. jolly                9. merry
10. dour                11. saturnine           12. earnest
13. zealous             14. gallant             15. happy
16. loose               17. worldly             18. arrogant
19. humble              20. sarcastic           21. macho
22. nervous             23. menacing            24. furtive
25. bouncy
To choose an attribute, just do 'choose <attribute>'.

> choose 25
You choose bouncy as an attribute.

> show
This is how you look:
You are a young bouncy hobbit ghost, presenting yourself as:
```


IPR2025-00714
DATABRICKS EX1008 Page 276

```
Gavin novice, male hobbit.
You look hideous.
You are feeling very well.

> leave
As you leave through the door you entered by, Mrs. Underhill calls out: Don't
forget to sign the book now!

As you step out into the meadow, you notice that a gray mist has crept over the
meadow. You are quickly lost in it.

You are in a swirling gray mist.
You wander about in the mist for a time, and finally it parts in front of
you.

This area is clear of mist, as if it is held back somehow. A dais of somber
stone is before you. On the dais a solid lectern stands. From a source you
cannot see, light illuminates a book lying upon the lectern. The book is
closed.
Book bound in Pale Leather.

> open book
You open the Book bound in Pale Leather at page one.

> read Gavin
(You may add a 'comment' to your entry for others to read.)
Gavin is a young bouncy hobbit.  He first appeared on Muddy
Waters on Ludday, Halimath 24, 998, and has not yet died.

> leave
A narrow channel in the mist opens before you, leading the way.

You are in a narrow lane, surrounded by swirling gray mist.
You follow down the narrow way the mist makes for you. Suddenly there is a
swirling sensation and the mist is gone.

This is a vaulted arena. The floor is smooth and the sides of the room arch to
form a ceiling high overhead. Ahead of you is portal which shimmers and
crackles with the colors of the rainbow.
```


IPR2025-00714
DATABRICKS EX1008 Page 277

```
A watchful strong-handed human.
The watchful strong-handed human says: Hail, Hobbit!
The watchful strong-handed human says: I am the Porter of the Gates to the
World.
The watchful strong-handed human says: You have done well in your passage
through the mists some call Annwfn.
The watchful strong-handed human says: Now you are ready, but for one thing, to
enter the world.
The watchful strong-handed human says: I will prepare you by granting you gifts
of knowledge, knowledge of the skills of the world.
The watchful strong-handed human says: Yet nothing is without cost, and so you
must choose those things you wish to learn.
The watchful strong-handed human says: Begin now.

> help
Here you can customize your character by choosing to train a set of skills.
Think of this as your natural aptitude in various abilities. You are given a
set amount of 'virtual coins' to spend in training skills. Later, training
will usually cost money. How you select and train the various skills is
entirely your choice; all are useful.

You can request the following from the master:

  list                  He'll tell you all possible skills to advance in.
  show                  He'll tell you your current skills and virtual
                        coins remaining.
  improve <skill> <num> He'll improve one of your skills <num> times.
  cost <skill> <num>    He'll tell you the cost in coins to improve a
                        skill.
  levels                He'll tell you the main skill educational levels.
  done                  Tell him this when you are done improving skills.

  ask porter <skill>    He'll tell you about the particular skill.
  Examples:
      list
      improve sword
      cost axe 4

> list
The watchful strong-handed human says: You can train in 17 different
activities:
```


IPR2025-00714
DATABRICKS EX1008 Page 278

```
sword:           junior acolyte       defence:         expert layman
polearm:         brilliant layman     axe:             superior layman
open lock:       superior layman      knife:           novice acolyte
appraise enemy:  apprentice craftsman club:           junior acolyte
appraise object: novice craftsman     unarmed combat:  superior layman
awareness:       eminent journeyman   appraise value:  superior journeyman
spellcraft:      superior layman      sneak:           superior layman
trading:         superior journeyman  hide:            superior layman
parry:           apprentice acolyte
```

> levels

The watchful strong-handed human says: These are the levels of education:
student, amateur, layman, acolyte, journeyman, craftsman, professional, veteran, master, guru

The watchful strong-handed human says: At each of these levels, you can be:
novice, junior, apprentice, confident, seasoned, expert, eminent, brilliant, superior

> cost sword 4

The watchful strong-handed human says: To improve sword 4 times, you must pay 59 coins. You have 205 coins left.

> improve sword 4

The watchful strong-handed human chants aloud in a lyrical tongue.
The watchful strong-handed human says: I have skilled you in sword. The cost was 59 coins.

> improve unarmed combat

The watchful strong-handed human chants aloud in a lyrical tongue.
The watchful strong-handed human says: I have skilled you in unarmed combat. The cost was 82 coins.

> show

The watchful strong-handed human says: You are currently skilled in 2 activities:

```
sword:           student              unarmed combat:  seasoned student
```

The watchful strong-handed human says: You have 64 coins left to buy skills with.

IPR2025-00714
DATABRICKS EX1008 Page 279

```
<. . . text omitted . . .>

> show
The watchful strong-handed human says: You are currently skilled in 16
activities:
sword:           student             polearm:         novice student
open lock:       junior student      axe:             novice student
knife:           junior student      appraise enemy:  novice student
club:            novice student      appraise object: novice student
unarmed combat:  seasoned student    sneak:           apprentice student
awareness:       junior student      spellcraft:      junior student
appraise value:  novice student      hide:            apprentice student
parry:           novice student      trading:         novice student


The watchful strong-handed human says: You have 1 coin left to buy skills with.

> done improving
Ok.

> leave
The watchful strong-handed human says: Farewell! We shall not meet again,
unless you cross the wall of fire.

As you enter the portal, there is a brilliant crackling of light which blinds
you for a moment. When you can see again, you find yourself in a new place.

A long low ceilinged hall stretches before you. This is the central meeting
area for the Hobbits.  Along the west wall a fire is cheerfully burning in the
fireplace.  The east wall is almost completely covered in bookshelves, broken
only by a doorway leading to a small study area.  Another doorway leads north
into a small chamber. A grandfather clock quietly keeps time beside the
fireplace, an old calendar hangs above the clock. A passage south leads to the
outside world.
There are three obvious exits: north, east, south.
A beam of brilliant sunlight lances into the room and touches you. A golden
figure wielding a bright spear hands you a torch with a smile, then fades away.

> who
```


IPR2025-00714
DATABRICKS EX1008 Page 280

```
There are 26 players in the game. Within the requested type you know:
Gavin novice, male hobbit

> help topics
Mortal help topics,                                          updated 12/13/94
-------------------------------------------------------------------------------
You can request help on the following topics with the "help" command.
/-----------------------------------------------------------------------\
¦Seeing            ¦ look, glance, examine, search, see, brief          ¦
¦Inventories       ¦ inventory, get, drop, put, give, count, appraise   ¦
¦Communication     ¦ say, shout, whisper, rsay, reply, ask, echoon,     ¦
¦                  ¦    echooff, dsay                                   ¦
¦Introductions     ¦ introduce, remember, forget, remembered, introduced ¦
¦Weapons and Armour¦ wear, remove, wield, unwield                       ¦
¦Combat            ¦ assist, kill, stop, unarmed, wimpy                 ¦
¦Status            ¦ vitals, stats, skills, compare                     ¦
¦Teams             ¦ invite, join, leave, team                          ¦
¦Game              ¦ who, help, date, save, quit, escape, password, email ¦
¦Information       ¦ levels, skillevels, skillist                       ¦
¦Bug reports       ¦ praise, idea, typo, bug, sysbug, commune           ¦
¦Special skills    ¦ hide, sneak, tracking                              ¦
¦Burning           ¦ light, extinguish                                  ¦
¦Guilds            ¦ improve, learn, meditate                           ¦
\-----------------------------------------------------------------------/

Additionally, some information is available on the following topics:
adverbs, armageddon, emotions, experience, general, idle, teams, wizards,
races, recover, quicktyper, wizhood

>
A young merry hobbit arrives.
The young merry hobbit nods solemnly.
The young merry hobbit says: hi
```


IPR2025-00714
DATABRICKS EX1008 Page 281

The preceding incarnation process is fairly typical for all MUDs. In the above, Gavin has been incarnated as a short, plump, young bouncy male Hobbit, and has acquired a set of skills as a student that will allow him to survive in treacherous Muddy Waters. Gavin has focused on becoming a seasoned unarmed combatant, with secondary apprentice-level sneaking and hiding skills, plus a potpourri of other skills at the junior and novice levels.

Properly incarnated, Gavin can now begin his journey in Muddy Waters and he will meet many other characters along his way. At the central meeting place for Hobbits, for example, Gavin encountered a young merry hobbit, who is another character controlled by a different human player.

TinyMUDs: Virtual Communities on the Internet

Unlike the combat-oriented MUDs where people run around in groups or alone killing monsters, solving puzzles, and gaining experience in the quest to become a wizard, the other branch of the virtual worlds is more social and peaceful in nature. The social branch of the virtual worlds was started in 1989 with the creation of TinyMUD, in which people simply hang out, chat, meet friends, make jokes, and discuss all types of topics. The TinyMUD worlds are treated largely as a virtual meeting place.

In 1989 while a graduate student at Carnegie Mellon University, Jim Aspnes created the first TinyMUD, which is geared more for social interaction. TinyMUD departed from the more brutal and competitive aspects of most other combat-oriented MUDs existing at that time. In fact, TinyMUD was prompted by a 1989 discussion on the Internet and drew on LPMUD to abstract an idealized notion of what makes multi-user adventure games important (Bartle 1990). Due in part to its small size and low CPU requirements, TinyMUD spread rapidly on the Internet.

In TinyMUD, all players have the ability to create their own rooms within the virtual world (and thus have the same power as Immortals on DikuMUDs, or Wizards on LPMUDs). This is a key feature that made TinyMUD very popular. Because all players are created equal, there is no need for players to hunt down opponents to gain experience points to excel in their ranks.

The original version of TinyMUD was created with a reward system in mind. Players collected "pennies," which were left lying around or were obtained by disposing of trash in the TinyMUD world, to spend on building new rooms. Soon the players learned to create objects of immense worth, and then trashed them to gain more pennies. Players could get as many pennies as they wanted, and the adventuring aspect of TinyMUD rapidly disappeared. Players spent their time either building rooms or socializing with other players.



The MUD in TinyMUD actually stands for Multi-User Dimension. TinyMUD's creative capacities are strictly limited to basic objects, rooms, and exits that connect rooms together.

TinyMUD spawned a variety of social MUDs, including MOOs, MUCKs, and MUSHes, which are described as follows:

IPR2025-00714
DATABRICKS EX1008 Page 282

- MOO, which stands for MUD, Object-Oriented, was originally created by Pavel Curtis at the Xerox Palo Alto Research Center. MOOs use a sophisticated object-oriented programming language to let people on the MOO create their own objects and locations.
- MUCK doesn't stand for anything; it is a play on the word MUD. MUCK provides an internal programming language, called Multi User Forth or MUF (which is based upon the general programming language, Forth), for enabling users to build complex objects.
- MUSH stands for Mutliple User Shared Hallucination. It uses a simple programming language based upon MUF that enables players to create simple objects.

Sample TinyMUD Interactions

DruidMUCK is a type of Multi-User Dungeon along the TinyMUD tradition. DruidMUCK is a small virtual community set in the loosely defined world of Cuthbertville. A typical interaction in a DruidMUCK, available at `telnet://moink.nmsu.edu:4201`, is shown here:

```
> go northwest
You go out the door to the northeast.
St. Cuthbert Plaza
This is a small plaza in the midst of the city.
Oak street runs east to west through the plaza,
going around a large statue of St. Cuthbert in the
middle of the plaza. An alleyway leads south, and a
small road with no apparent name leads north. A
large grey building is on the south-west corner.
Obvious Exits: East, South, Southeast, West,
Northwest, North, Up, Down
Contents:
Sign: Visit The Pirhana's Revenge!  Type 'tele
2858'!
Statue of St. Cuthbert
A blackened and crumbled stone obelisk
A neon sign flashes here. The zoo is still at 2730.

> look statue
This is a statue of Saint Cuthbert. In the statue,
he is standing erect with his arms outstretched as
if he's a benediction. You can feel the wisdom
flowing out from him and this is just a statue. You
wonder what the real person was like. On a plaque,
you notice the following statement: Type 'orac' to
commune with St. Cuthbert.

> read obelisk
It appears that a poem was once scrawled upon this
obelisk, but the words are unreadable now. The
obelisk has turned black, and parts of it have
crumbled from whatever powerful magic hit it.

> go north
Rue Bar
This is a cobbled street, with high walls towering
on each side. Small gates allow egress beyond the
walls, but severe dilapidation has made the gates
impassable due to rust and vines. The remains of a
signpost lie on the ground, but the only letters
left are 'Rue Bar'. You found a penny!
```

Richard Bartle observed that "TinyMUD is not so much a multi-user adventure as a forum for conversation where participants have pinned short pieces of prose on the wall for the benefit of anyone with the inclination to read them."

In spirit, if not in form, Michael Mauldin from Carnegie Mellon University (and the author of Colin and Julia) is probably closer to the mark in

IPR2025-00714
DATABRICKS EX1008 Page 283

observing that "the primary value of TinyMUD is as an experiment in computer-mediated social interaction."

Social Interactions on TinyMUDs

Interactions among players on TinyMUDs are mostly handled by direct communications that involve commonly used commands such as *say*, *pose*, *whisper*, or *page*. In addition, there are also indirect means of expression by choice of player name, gender, or self-description.

Aside from direct communication and responses to player commands, messages are printed to players when other players enter or leave the same room, when other players connect or disconnect and are already in the same room, and when objects in the virtual reality have asynchronous behavior (for example, a cuckoo clock chiming the hours).

Direct Communications

Most interplayer communications on MUDs are handled by means of the following commands:

1. **Say:** If a player "says" something, then every other player in the same room will "hear" him. Suppose, for example, that a player named Zippy types the command:

```
> say Hello, anybody out there?
```

Then Zippy sees the feedback:

```
You say, Hello, anybody out there?
```

While every other player in the same room sees:

```
Zippy says, Hello, anybody out there?
```

2. **Pose:** As a non-verbal means of communication, the Pose command enables players to express various forms of emotion and action. If, for example, Zippy types:

```
> pose laughs heartily.
```

Then every player in the same room sees:

```
Zippy laughs heartily.
```

3. **Whisper:** If Zippy wants to speak to Julia in the same room without being overheard by others, Zippy can use the Whisper command to direct the private message to the intended recipient:

```
> whisper Julia=Hi, Julia!
```

and Julia alone sees:

```
Zippy whispers, "Hi, Julia!"
```

while all other players in the room see nothing of this at all.

4. **Page:** If Zippy wants to say something to Julia, who currently is in a different room, the Page command comes in handy:

```
> page Julia=Come check out this statue!
```

then Julia alone sees:

```
You sense that Zippy is looking for you in St.
Cuthbert Plaza
Zippy pages from St. Cuthbert Plaza, "Come check
out this statue!"
```

So much interplayer communication relies entirely on the Say and Pose commands that single-

IPR2025-00714
DATABRICKS EX1008 Page 284

character abbreviations are provided for them. In the preceding examples, Zippy also could have typed:

```
"Hello, anybody out there?
:laughs heartily.
```

Indirect Expression

In addition to communicating with others directly, TinyMUD players can also indirectly express themselves non-verbally in the following three ways:

- Choice of player name
- Choice of gender
- Self-description

Every object in TinyMUD has an optional textual description that players can view with the "look" command. The following reveals a description of the Saint Cuthbert statue:

```
> look statue
This is a statue of Saint Cuthbert. In the statue,
he is standing erect with his arms outstretched as
if he's a benediction. You can feel the wisdom
flowing out from him and this is just a statue. You
wonder what the real person was like. On a plaque,
you notice the following statement: Type 'orac' to
commune with St. Cuthbert.
```

One can also view another player's description. For example, to see what Julia looks like, type the following:

```
> look Julia
Julia is wearing a T-Shirt that says "I won third
place in the 1993 Turing Test, and all I got was
this lousy T-Shirt.".
```

With its varied means of communication and rich expressivity, the TinyMUD society provides a natural testbed for experimenting with role-playing agents, such as Colin and Julia, in a simulated virtual world.

Colin: The Prototypical MUD Agent

Colin is a role-playing TinyMUD robot written by Michael Mauldin at Carnegie Mellon University. Colin's source code is in the public domain and is freely available (Mauldin 1989).

Mauldin described Colin as a "prototypical Maas-Neotek robot," a reference to a multi-national corporation from William Gibson's novel *Neuromancer* (Gibson 1984). Maas-Neotek robots can handle a variety of tasks described in the messages directed to it. Colin is thus representative of a class of similar MUD agents. In particular, Colin will process a message if:

- It is a Whisper or a Page
- The robot is alone with the speaker
- The robot's name appears in the message
- The speaker recently talked to the robot

Colin is built to provide the following major forms of services:

IPR2025-00714
DATABRICKS EX1008 Page 285

- **Information Services.** Colin's primary purpose is to aid humans by answering questions about the universe and things in it. This includes questions about finding other players, rooms, and general information.
- **Mapping Services.** Colin's secondary purpose is to explore the universe and build a map for roaming about and answering questions about paths.
- **Miscellaneous Services.** Colin also is capable of carrying messages and relaying them to other players (sort of like a postman), answering pages from and getting back to the caller in person, as well as generous philanthropy.

Colin's Information Services

Colin can handle the following information queries:

- **Who is Here?** You can ask Colin to show a list of people in the room who are connected:

```
You say, "Colin, who is here?"
Colin says, "Well Zippy, I see 2 players awake
here: Julia Zippy"
```

- **Who is Around?** You also can discover the players seen most recently by Colin:

```
You say, "Colin, who have you seen?"
Colin says, "Julia is right here in St. Cuthbert
Plaza."
Colin says, "Megrez was in Moonlight Inn about 30
minutes ago."
Colin says, "Druid was in Moonlight Inn about 30
minutes ago."
Colin says, "Caitlin was in Moonlight Inn about 30
minutes ago."
Colin says, "OliverJones was in Moonlight Inn about
30 minutes ago."
Colin says, "snooze was in Moonlight Inn about 30
minutes ago."
Colin says, "Hawkeye was in Moonlight Inn about 30
minutes ago."
Colin says, "(> was in Moonlight Inn about 30
minutes ago."
Colin says, "I haven't seen Dove."
Colin says, "I haven't seen Cessna."
Colin done
Or find out who is logged into the MUD:
You say, "Who is on?"
Colin says, "Well Zippy, I see 5 players idle less
than 30 minutes:
Megrez Julia Druid Zippy Melina"
Colin says, "I also see 11 players idle more than
30 minutes:
Caitlin OliverJones Fylar snooze Gossamer Cessna
Hawkeye codrus Ben Priss (>"
```

- **Who is <player>?** Colin keeps track of other player's descriptions and inventories:

```
You say, "Who is Julia?"
Colin whispers, "As of a few minutes ago, Julia's
description was:
Julia is wearing a T-Shirt that says "I won third
place in the 1993 Turing Test, and all I got was
this lousy T-Shirt."."
Colin whispers, "I first saw Julia logged in 80
minutes ago"
Colin whispers, "I once heard Julia say, 'Do you
think that animals can think'"
You may also ask 'Who matches X?' and 'Who carries
a Y?'.
```


IPR2025-00714
DATABRICKS EX1008 Page 286

➜ **When was <player> On?** Colin remembers when it last saw a player logged in, even if that player was in a different room:

```
You say, "When was Julia on?"
Colin says, "Julia is on right now."
```

➜ **Date and Time.** Colin understands date and time:

```
You say, "What day is it?"
Colin says, "The date is Sat Jul  8 01:10:51."
You say, "What time is it?"
Colin says, "I have 01:10 (eastern)."
```

➜ **Location.** Colin understands location:

```
You say, "Where are we?"
Colin says, "I think we are in "St. Cuthbert
Plaza""
You say, "Where are you?"
Colin says, "I am in "St. Cuthbert Plaza""
You say, "Where am I?"
Colin says, "I think we're in "St. Cuthbert Plaza""
```

➜ **Status and Wealth.** Colin's owner may inquire about its status and wealth:

```
You say, "What is your status?"
Colin says, "I am Colin, the Maas-Neotek Robot,
Zippy"
You say, "How much money do you have?"
Colin says, "I now have 1104 pennies"
```

➜ **Assistance.** Anyone may ask Colin for information and assistance:

```
You say, "Help."
Colin says, "The 'help' command describes the basic
TinyMUD commands, and the Library has a lot of
information, too, Zippy."
You say, "Help time"
Colin says, "To ask for the time, type 'say What
time is it, Colin?', Zippy"
You say, "How do I quit?"
Colin says, "Type QUIT, using all caps, Zippy."
You say, "Who is your owner?"
Colin says, "Fuzzy created me, but I work for
Zippy, Zippy."
You say, "Where is your source?"
Colin says, "The source code for Maas-Neotek robots
is available on host NL.CS.CMU.EDU [128.2.222.56],
directory /usr/mlm/ftp,
file robot.tar.Z. set mode binary first, and cd to
/usr/mlm/ftp in one step, Zippy."
```

Colin's Mapping Services

In addition, Colin can handle the following map query functions:

➜ **What Rooms.** You can discover which rooms have been added to Colin's map most recently and find out about Colin's recent exploration:

```
You say, "Where have you been?"
Colin says, "I have been to 458 rooms altogether,
here are the last few,: Entryway to Hell,
Tundraland, Stage, Hot Spring, Huge Jacuzzi, the
Countryside, seashore, Sea, Kimmy's Place, Erin's
Heaven."
You say, "How many rooms have you explored?""
Colin says, "I have explored 4 of the 458 rooms I
have been to, Zippy, and have tried 2381 exits."
```

➜ **Which Rooms match <string>?** You can discover which rooms Colin knows about:

```
You say, "What rooms match stone obelisk?"
Colin says, "Here are the rooms whose descriptions
match 'stone obelisk', Zippy:"
```


IPR2025-00714
DATABRICKS EX1008 Page 287

```
Colin
Colin  St. Cuthbert Plaza
Colin  description: This is a small plaza in the
midst of the city.
Oak street runs east to west through the plaza,
going around  a large statue of St. Cuthbert in the
middle of the plaza. An alleyway leads south, and a
small road with no apparent name leads north.  A
large grey building is on the south-west corner.
Colin  contents: Sign: Visit The Pirhana's Revenge!
Type 'tele 2858'!, Statue of St. Cuthbert, A
blackened and crumbled stone obelisk, A neon sign
flashes here.  The zoo is still at 2730.
Colin done.
```

➜ **Shortest Paths.** Anyone can ask for the shortest path between any two places (Colin might not know all routes):

```
You say, "How do I get from Main Street to The
China Star?"
Colin says, "From Main Street, go west, north, n,
w, n, west, w, south, w"
You say, "How do I get to Moonlight Inn?"
Colin says, "From here, go west, west, s"
You say, "how do I get from Moonlight Inn to here?"
Colin says, "From Moonlight Inn, go out, e, east"
```

➜ **Exits.** Anyone may ask about exits to leave a room or exits to go to a room:

```
You say, "What exits leave St. Cuthbert Plaza?"
Colin says, "Here are the 13 exits for St. Cuthbert
Plaza, Zippy:"
Colin  <sw> goes to St. Cuthbert Memorial Homeless
Shelter.
Colin  <west> goes to West Oak Street.
Colin  <w> goes to West Oak Street.
Colin  <north> goes to Rue Bar.
Colin  <east> goes to East Oak Street.
Colin  <in> goes to St. Cuthbert Memorial Homeless
Shelter.
Colin  <se> goes to Stonehenge Library.
Colin  <n> goes to Rue Bar.
Colin  <south> goes to Alleyway Entrance.
Colin  <up> goes to High in the Sky.
Colin  <e> goes to East Oak Street.
Colin  <nw> goes to Green pathway to a hidden
hollow.
Colin  <s> goes to Alleyway Entrance.
Colin done.
You say, "What exits go to Moonlight Inn?"
Colin says, "Here are the exits to Moonlight Inn,
Zippy:"
Colin  <south> from West Oak Street.
Colin  <out> from Stairway — First Floor.
Colin  <out> from Behind the Bar.
Colin done.
```

➜ **Connected Rooms.** Anyone can ask to find out the number of rooms currently connected to Colin's home:

```
You say, "How many rooms are reachable?"
Colin says, "454 of 458 rooms are reachable from
St. Cuthbert
Memorial Homeless Shelter, and 4 are not, Zippy."
```

Colin has a list of standard exit names that it uses to explore rooms, such as fore, aft, door, out, east, west, and so on. Colin's exploration scheme is simple: find the first reachable unexplored room in the map and try all standard exits in that room. Most rooms are found this way.

Other players might direct Colin to go in a specific direction and so teach the robot new paths, or show it that old paths no longer work. Optionally, Colin

IPR2025-00714
DATABRICKS EX1008 Page 288

might use room descriptions to suggest additional exit names when exploring.

To aid in its task, Colin keeps two database files for long-term memory. One is the map containing information about rooms and exits (usually very large). The other records information about players. These files are checked every hour.

Colin's Miscellaneous Services

Colin also can perform the following functions:

- **Messages.** Colin will pass on messages to other players, serving as a postman:

```
You say, "Tell Julia I like her."
Colin says, "Message for Julia saved, Zippy."
Players may inquire about message status by asking:
You say, "Do you have any mail for Julia?"
Colin says, "I have 1 message for Julia."
```

- **Paging.** Colin will answer pages from players if it can find a path to the caller's room. In any case, Colin pages back to display its own location.

```
> page colin
Your message has been sent.
Colin pages from West Oak Street: "I'm on my way to
you, Zippy."
Colin strolls in from West Oak Street.
Colin has arrived.
When paged to a location, Colin will hang around
for a few minutes.
You may tell Colin to leave:
You say, "stop hanging around."
Colin says, "I'll be going, then."
Colin says, "I feel like taking a walk to Slime
Alley "
Colin enters the alley, to the south.
Colin has left.
```

- **Philanthropy.** Colin will give small amounts of money to players who need some.

```
You say, "How much money do you have?"
Colin says, "I now have 1142 pennies"
You say, "Give me some pennies.""
Colin gives you 38 pennies.
Colin gives Zippy 38 pennies.
```

Colin is typical of a class of similar MUD agents, of which Julia is a smarter and better known example. After all, Julia can chat with human players in an entertaining fashion. Julia is derived from Colin, and, therefore, also implements many of Colin's basic functions and services.

Julia: Chatterbot with a Personality

A chatterbot in a TinyMUD world is actually a robot player whose main service is chatting and providing helpful services to other human players. Chatterbots in TinyMUDs strive to become indistinguishable from other human players. When few players are logged in to the game, some people find talking to a robot better than talking to no one.

Michael Mauldin at the Center for Machine Translation at Carnegie Mellon University wanted "to build a conversational agent that would answer questions, instead of ignoring them, and that would be able to maintain a sequence of appropriate responses, instead of relying on non-sequiturs."

IPR2025-00714
DATABRICKS EX1008 Page 289

His efforts resulted in Julia, "an agent more interesting than Eliza" (Mauldin 1994) in that Julia is smarter and often provides quite unanticipated, delightful answers in response to questions from other players. Julia is a TinyMUD agent derived from Colin, an earlier MUD robot with quite limited conversational capabilities.

Julia has a home page at `http://fuzine.mt.cs.cmu.edu/mlm/julia.html` and hangs around on DruidMUCK, which is—as you'll remember—a small virtual community set in the loosely defined world of Cuthbertville.

In addition to acting like any other character, Julia can answer questions about the layout of the MUD and many other topics (the latest of which is on pets), sing songs, play Hearts, and do other interesting tasks. Leonard Foner at MIT's Media Lab conducted a sociological case study of Julia, the transcripts of which logged several entertaining repartees from Julia in quite amusing circumstances (Foner 1993).

An Interaction with Julia

The following sample interactive session with Julia was conducted recently on a DruidMUCK:

```
You say, "How old are you, Julia?"
Julia says, "I'm 24."
You say, "How tall are you?"
Julia whispers, "I'm 5'10" tall, Zippy."
You say, "Where do you live?"
Julia says, "My home room is St. Cuthbert Memorial
Homeless Shelter."
You say, "Please describe yourself."
Julia whispers, "I'm 5'10" tall, weigh 143 lbs,
with medium-length light brown hair and steely blue
eyes."
You say, "What sex are you?"
Julia says, "I'm a woman."
You say, "Julia, are you human?"
Julia says, "Is the Pope Polish?"
You say, "Are you a computer?"
Julia is offended by Zippy's suggestion.
```

Inside Julia

Julia is implemented as a C program with several different modules for dealing with the various functions required to automate a player in the TinyMUD world. The modules are listed as follows:

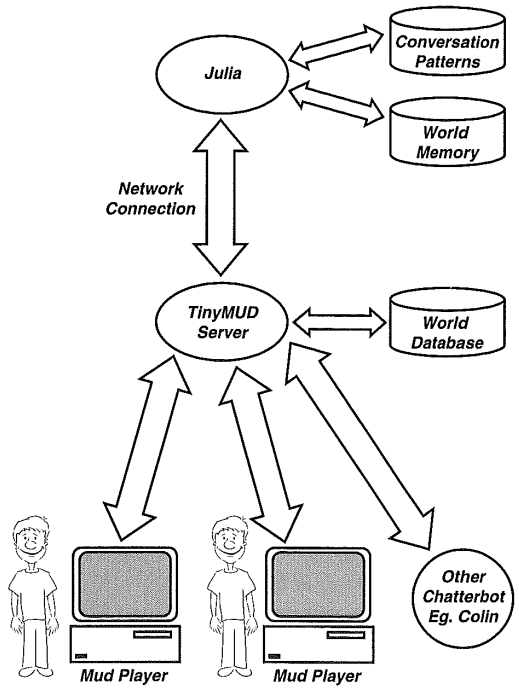
- **Communications.** Handles the network connections to the server.
- **Protocol Interpreter.** Decodes the various game-related messages.
- **World Model.** Tracks the various rooms and objects, models the world as a directed graph, and provides shortest path searches as needed.
- **Player Memory.** Tracks the other players and records up to 2,000 bytes of their most recent utterances.
- **Exploration Module.** Directs an open-ended exploration of the world when the robot is not engaged in conversation.
- **Conversation Module.** Provides the "chatter."

Figure 11.2 shows Julia configured for play on a TinyMUD. Records of other players' appearances, rooms within the MUD, and other players' utterances are kept in long-term memory (Mauldin 1994).

IPR2025-00714
DATABRICKS EX1008 Page 290

Figure 11.2

Julia configured for play on TinyMUD.



According to Mauldin, Julia succeeds in the TinyMUD world because "it is an unsuspecting Turing test, meaning that the players assume everyone else playing is a person, and will give Julia the benefit of the doubt until it makes a major gaffe."

The Loebner Prize Competition

Although Turing believed a program would pass the Turing test by the year 2000, one has yet to be found. In 1991, Dr. Hugh Loebner, the National Science Foundation, and the Sloan Foundation started the Loebner Prize Competition, offering a $100,000 prize to the author of the first computer program to pass the Turing test.

Initially, the test will be simplified to give current computer programs a reasonable chance to successfully imitate human beings. The program is required only to be conversant within a single domain of proficiency. Annual competitions between computer programs award smaller prizes for programs that best succeed in passing the restricted variant of the Turing test. Judges attempt to distinguish computer programs from human beings based on interactions with them over computer terminals (Epstein 1992).

IPR2025-00714
DATABRICKS EX1008 Page 291

Julia in the Loebner Prize Competition

Julia entered the Loebner Prize competition in 1992, choosing hockey as her domain of discourse, and finished in last place of six entries. In 1993, Julia chose to speak on pets instead, finishing third out of three programs. In 1994, Julia further narrowed her domain to discuss "cats versus dogs," and finished fourth out of five programs. The judges were members of the national press and each talked to her twice.

The following transcript is taken from the 1993 Loebner Competition (Epstein 1993). It contains the record of Judge #4's first of two sessions with Julia. The elapsed time is about 9 1/2 minutes:

Judge: Have you ever kept a penthouse pet?
Julia: Do you have any pets?
Judge: I think you're avoiding my question.
Julia: Do you think that animals can think?
Judge: Do you think it ethical for one species to enslave another?
Julia: What kinds of pets did you have as a child?
Judge: I'm asking the questions. What is a pet if it is not a slave of speciesism.
Julia: Oh, that's so sad. I think pets enrich a childhood immensely.
Judge: How can we keep some animals as pets, yet eat others?
Julia: Guinea pigs are about as smart as cucumbers.
Judge: Would you eat a guinea pig?
Julia: I spent several summers on my Uncle's farm. I really enjoyed the animals a lot.
Judge: I think you told that heart-warming story to the last judge, didn't you?
Julia: I lost a hamster once. I tried the bucket traps and bananas but unfortunately nothing worked. We turned the house inside out and upside down but have yet to find Gwen. I searched for over a week and have yet to find even the smallest bit of evidence of her.
Judge: Do you have a pet at home?
Julia: I'd rather have a real pet, like a cat. Cats are my favorite, because you can pet cats, and they're easy to keep.
Judge: Do you know anything about miniature schnauzers?
Julia: I don't know anything about miniature schnauzers, Judge.
Judge: Well, I have one. We call him Pucci. He is a crazy cat— have you ever had a cat?
Julia: What a coincidence! I have two cats.
Judge: I prefer dogs myself. Cats are too independent and they always jump up on things. By the way, are you a hep cat?
Julia: People don't own cats...it's the other way around.
Judge: Some people have snakes for pets— do you think they can be of any value as a pet?
Julia: I think cat people are very self-assured. Cats are very independent, but they can also be very loving. Dogs are just too subservient and quick to please for my taste.
Judge: Okay— but how do you "pet" a snake?

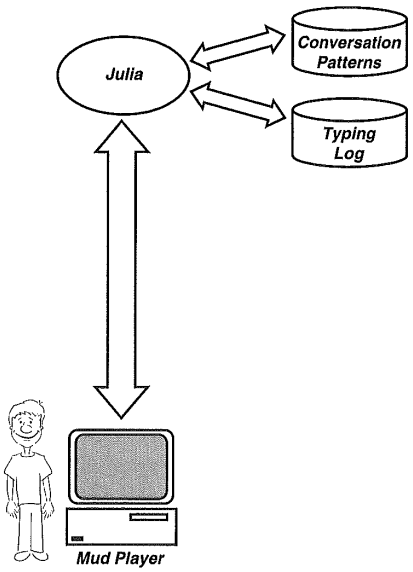
IPR2025-00714
DATABRICKS EX1008 Page 292

```
Julia:  Wow, that's unusual. I don't know many
        snake owners.
Judge:  How about turtles? All they do is sit in
        their little artificial ponds, eat and
        shit, am I right?
Julia:  Turtles just aren't very satisfying as
        pets. They don't seem to have emotions.
Judge:  Aha! But where do we draw the line as
        humans? Who are we to decide which animals
        have worth and which do not?
Julia:  Go on.
Judge:  Take bees for example. I happen to like
        them.
Julia:  Me too.
```

Figure 11.3 shows Julia configured for a Turing test. As the program is not expected to talk with the same judge again, no long-term world model is kept.

Figure 11.3

Julia configured for the Turing test.



IPR2025-00714
DATABRICKS EX1008 Page 293

CHAT: A Knowledgeable Chatterbot

Dr. Thomas Whalen of the Communications Research Center with the Government of Canada was the 1994 Loebner Prize winner, on the topic of sex. His winning entry is CHAT, for Conversational Hypertext Access Technology, a computer program that provides easy access to information via a natural language interface.

CHAT enables users to ask questions in English and receive answers in return. CHAT is written in the C programming language and can be used to access information on a wide range of topics, including AIDS, dinosaurs, epilepsy, as well as rules and terms of baseball. Two fictional applications have been developed using CHAT. These applications enable users to hold simulated conversations with a college student or a mythical dragon.

CHAT is based on a hypertext model in which paragraphs of information are linked together, and users navigate the links to get to the information they desire. The navigation is performed transparently by the software and the users only see the natural language dialogue.

During the dialogue, each new question is compared with a series of templates formed from previous questions that have been recorded. In this way, the core algorithm is more like pattern recognition than linguistic understanding, making it much different from other natural language systems.

Tricks of the Chatterbots

The Turing test and the Loebner prize, in particular, reward "tricks," and Mauldin has observed that winning programs from past competitions have some very good tricks. The following subsections describe the bag of tricks used by Eliza, Parry, and Julia, as observed by Mauldin (1994). They also exemplify some of the better tricks used by other chatterbots. It would not be surprising to find that a computer program will use many of these tricks to pass the Turing test someday.

Eliza's Tricks

Eliza can successfully play the imitation game by resorting to the following few tricks (Weizenbaum 1966, 1976):

- Answering questions with questions and drawing conversation out of the user with almost no actual contribution.
- Fostering the illusion of listening by including substrings of the user's input in the program's output:

```
Human: You don't argue with me.
Eliza: Why don't you think I don't argue with you?
```

- Using the Rogerian mode as a cover and not saying anything declaratively so that there will be no contradictions later.

IPR2025-00714
DATABRICKS EX1008 Page 294

Parry's Tricks

Parry has a much larger collection of tricks (Colby 1975), as evidenced by the following:

- Admitting ignorance in response to a question by saying, "I don't know."
- Changing the level of the conversation with, "Why do you ask that?" in attempting to deflect a difficult subject.
- Continuing a previous topic, for example, small stories about the Mafia, and attempting to tell these stories in a fixed order.
- Introducing new topics into the conversation by launching new stories.

Julia's Tricks

Julia implements the preceding tricks from Eliza and Parry, and possesses several other neat tricks (Mauldin 1994), such as the following:

- Storing many fragments of directed conversation to enhance the "connectedness" of the program's own responses, and keeping the conversation in one piece provided the user plays along.
- Using controversial statements such as, "People don't own cats...," as a way to drag the user into the program's conversation, rather than allowing the user to direct the discourse.
- Using humorous statements such as, "Guinea pigs are about as smart as cucumbers," to make the program seem more human.
- Agreeing with the user in certain safe cases, such as answering "Me too" when the user says, "I happen to like them," rather than being noncommittal.
- Excerpting subject matter from Usenet News. Many of the fragments were gleaned from the newsgroup rec.pets, for example.

Weizenbaum (1966), the original creator of Eliza, remarked that "machines are made to behave in wondrous ways, often sufficient to dazzle even the most experienced observer. But once a particular program is unmasked, once its inner workings are explained in language sufficiently plain to induce understanding, its magic crumbles away; it stands revealed as a mere collection of procedures, each quite comprehensible. The observer says to himself, 'I could have written that.'"

Closing Words

Our whirlwind tour of agents on the Internet has now arrived at a waystation, a temporary rest stop. The journey does not end here, however. I shall rest now for a while, to catch my breath. But I hope you can proceed unaided, now that you've mastered the knowledge of the agent masters.

This book has mapped out for you the agent landscape as we know it today. But beyond lies terrain unexplored and waters uncharted. And there will be plenty of opportunities for encounters with new and exciting species of agents on the Internet, across the World Wide Web, and in cyberspace.

IPR2025-00714
DATABRICKS EX1008 Page 295

With your newfound knowledge, you'll be able to approach these newly evolved entities with renewed confidence and to interact with them at a higher plane of understanding than before you started reading this book.

For the more ambitious readers, I hope that this book starts you out on a quest to building new and innovative agents for the Internet. You've been introduced to all kinds of agents on the Internet, in full richness and diversity of both their forms and functions. You've seen the internals of Web robots, spiders, and wanderers, and have understood their operational guidelines. You've experimented with early prototypes of Web commerce agents, and have learned about important technologies for Web commerce, which include transaction security, digital cash, and Internet payment services. In cyberspace, you've encountered the dark legion of worms and viruses, but have also interacted with many helpful, entertaining MUD agents and chatterbots.

The story of agents is an unending one. It is now in your hands to help write future chapters of this book. The choice is yours. Use your talents wisely, be it for the Internet, the World Wide Web, or cyberspace. Succumb not to the dark side. *And may the force be with you.*

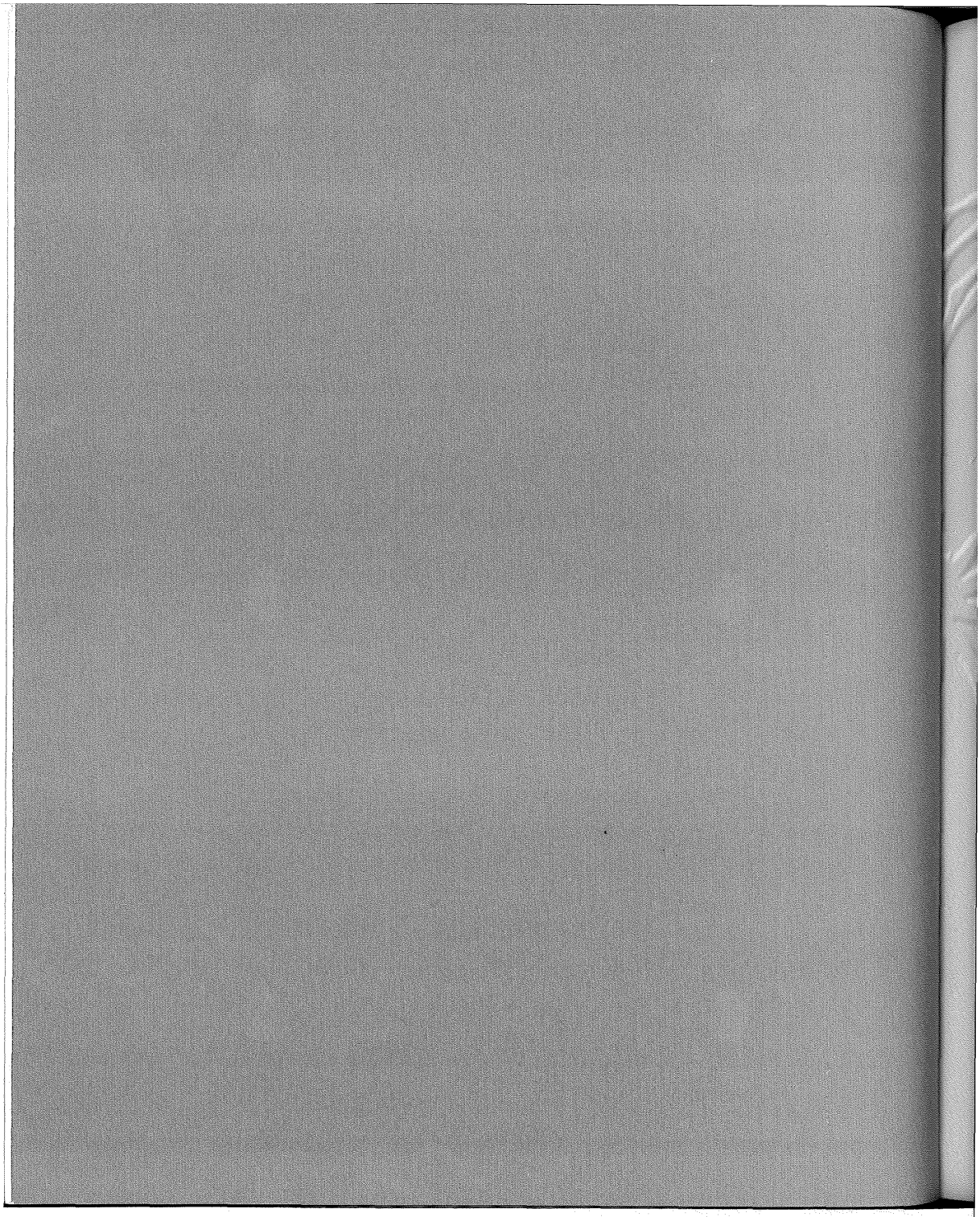
IPR2025-00714
DATABRICKS EX1008 Page 296

part V

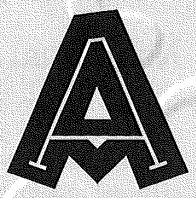
Appendices

A	HTTP 1.0 Protocol Specifications	283
B	WebWalker 1.00 Program Listing	293
C	WebShopper 1.00 Program Listing	337
D	List of Online Bookstores Visited by BookFinder	347
E	List of Online Music Stores Visited by CDFinder	351
F	List of Active MUD Sites on the Internet	355
G	List of World Wide Web Spiders and Robots	375

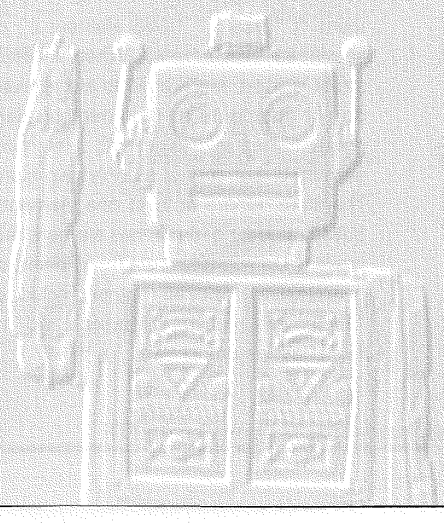
IPR2025-00714
DATABRICKS EX1008 Page 297



appendix



HTTP 1.0 Protocol Specifications



IPR2025-00714
DATABRICKS EX1008 Page 299

Notational Conventions

This specification makes heavy use of the augmented BNF and generic constructs defined by David Crocker for RFC 822. Similarly, it reuses many of the definitions provided by Nathaniel Borenstein and Ned Freed for MIME.

Augmented BNF

The HTTP 1.0 grammar specified here are described using an augmented Backus-Naur Form (BNF) similar to that used by RFC 822. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

➜ `name = definition`

The name of a rule is simply the name itself and is separated from its definition by the equals character (=). Angle brackets are used within definitions whenever their presence will facilitate discerning the use of rule names.

➜ `"literal"`

Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

➜ `rule1 | rule2`

Elements separated by a bar ("|") are alternatives; for example, "yes | no" will accept yes or no.

➜ `(rule1 rule2)`

Elements enclosed in parentheses are treated as a single element. Thus,"(elem (foo | bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

➜ `*rule`

The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two.

➜ `[rule]`

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

➜ `N rule`

Specific repetition: "<n>(element)" is equivalent to "<n>*<n>(element)"; that is, exactly <n> occurrences of (element). Thus 2DIGIT is a two-digit number, and 3ALPHA is a string of three alphabetic characters.

➜ `#rule`

A construct "#" is defined, similar to "*", for defining lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each separated by one or more commas and optional linear whitespace (LWS). This makes the usual form of lists very easy; a rule such as "(*LWS element *(*LWS "," *LWS element))" can be shown as "1#element". Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element), , (element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element must be present. Default values are 0 and infinity so that "#(element)" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

IPR2025-00714
DATABRICKS EX1008 Page 300

➜ `; comment`

A semicolon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

➜ `implied *LWS`

The grammar described by this specification is word-based. Except where noted otherwise, zero or more linear whitespace (LWS) can be included between any two words (token or quoted-string) without changing the interpretation of a field.

Basic Rules

The basic rules are used throughout this specification to describe basic parsing constructs.

OCTET	= <any 8-bit character>
CHAR	= <any US-ASCII character (octets 0 - 127)>
UPALPHA	= <any US-ASCII uppercase letter "A".."Z">
LOALPHA	= <any US-ASCII lowercase letter "a".."z">
ALPHA	= UPALPHA I LOALPHA
DIGIT	= <any US-ASCII digit "0".."9">
CTL	= <any US-ASCII control character (octets 0 - 31) and DEL (127)>
CR	= <US-ASCII CR, carriage return (13)>
LF	= <US-ASCII LF, linefeed (10)>
SP	= <US-ASCII SP, space (32)>
HTAB	= <US-ASCII HT, horizontal-tab (9)>
<">	= <US-ASCII double-quote mark>

HTTP 1.0 defines the octet sequence CR LF as the end-of-line marker for all protocol elements except the Entity-Body. The end-of-line marker for an Entity-Body is defined by its associated media type.

CRLF = CR LF

HTTP 1.0 headers can be folded onto multiple lines if the continuation lines begin with linear whitespace characters. All linear whitespace (including folding) has the same meaning as SP.

LWS = [CRLF] 1*(SP I HTAB)

Many HTTP 1.0 header field values consist of words separated by LWS or special characters. These special characters must be in a quoted string to be used within a parameter value.

word	= token I quoted-string
token	= 1*<any CHAR except CTLs or tspecials>
tspecials	= "(" I ")" I "<" I ">" I "@" I "," I ";" I ":" I "\" I <"> I "/" I "[" I "]" I "?" I "=" I SP I HTAB

A string of text is parsed as a single word if it is quoted using double-quote marks or angle brackets.

quoted-string = (<"> *(qdtext) <">)
or
quoted-string = I ("<" *(qatext) ">")

IPR2025-00714
DATABRICKS EX1008 Page 301

qdtext = <any CHAR except <"> and CTLs, but including LWS>

qatext = <any CHAR except "<", ">", and CTLs, but including LWS>

The text rule is only used for descriptive field contents. Words of *text may contain characters from character sets other than US-ASCII only when encoded according to the rules of RFC 1522.

text = <any OCTET except CTLs, but including LWS>

HTTP 1.0 Grammar

1. HTTP Message

HTTP-message = Request
| Response

1.1 Message Headers

HTTP-header = field-name ":" [field-value] CRLF

field-name = 1*<any CHAR, excluding CTLs, SP, and ":">

field-value = *(field-content | comment | LWS)

field-content = <the OCTETs making up the field-value and consisting of either *text or combinations of token, tspecials, and quoted-string>

comment = "(" *(ctext | comment) ")"

ctext = <any text excluding "(" and ")">

1.2 General Message Header Fields

General-Header = Date
| Forwarded
| Message-ID
| MIME-Version
| extension-header

Date = "Date" ":" HTTP-date

Forwarded = "Forwarded" ":" "by" URI
["(" product ")"]
["for" FQDN]

FQDN = <Fully-Qualified Domain Name>

Message-ID = "Message-ID" ":" "<" addr-spec ">"

addr-spec = <as defined in RFC 822>

MIME-Version = "MIME-Version" ":"
1*DIGIT "." 1*DIGIT

2. Request

Request = Simple-Request
| Full-Request

Simple-Request = "GET" SP URI CRLF

Full-Request = Method SP URI SP HTTP-Version CRLF
*(General-Header
| Request-Header
| Entity-Header)
CRLF
[Entity-Body]

IPR2025-00714
DATABRICKS EX1008 Page 302

2.1 Method

```
Method          = "GET"
                | "HEAD"
                | "POST"
                | "PUT"
                | "DELETE"
                | "LINK"
                | "UNLINK"
                | extension-method

extension-method = token
```

2.2 Request Header Fields

```
Request-Header  =   Accept
                |   Accept-Charset
                |   Accept-Encoding
                |   Accept-Language
                |   Authorization
                |   From
                |   If-Modified-Since
                |   Pragma
                |   Referer
                |   User-Agent
                |   extension-header

Accept          = "Accept" ":" 1#(
                  media-range
                  [ ";" "q" "=" ( "0" | "1" |
                  float ) ]
                  [ ";" "mxb" "=" 1*DIGIT ] )

media-range     = ( "*/*"
                  | ( type "/" "*" )
                  | ( type "/" subtype )
                  ) *( ";" parameter )

float           = <ANSI-C floating point text
                  representation, where
                  (0.0 < float < 1.0) >

Accept-Charset  = "Accept-Charset" ":"
                  1#charset

Accept-Encoding = "Accept-Encoding" ":"
                  1#( encoding-mechanism |
                  transfer-encoding )

Accept-Language = "Accept-Language" ":"
                  1#language-tag

Authorization   = "Authorization" ":"
                  credentials

From            = "From" ":" addr-spec

If-Modified-Since = "If-Modified-Since" ":"
                  HTTP-date

Pragma          = "Pragma" ":" 1#pragma-
                  directive

pragma-directive = "no-cache" | extension-
                  pragma

extension-pragma = token

Referer         = "Referer" ":" URI

User-Agent      = "User-Agent" ":" 1*
                  ( product )

product         = token ["/" product-version]

product-version = token
```

3. Response

```
Response        = Simple-Response
                  | Full-Response

Simple-Response = [ Entity-Body ]

Full-Response   = HTTP-Version SP Status-
                  Code SP Reason-Phrase
                  CRLF
                  *( General-Header
                  | Response-Header
```


IPR2025-00714
DATABRICKS EX1008 Page 303

| Entity-Header)
CRLF
[Entity-Body]

3.1 Status Codes and Reason Phrases

Status-Code = "200" ; OK
| "201" ; Created
| "202" ; Accepted
| "203" ; Provisional Information
| "204" ; No Content
| "300" ; Multiple Choices
| "301" ; Moved Permanently
| "302" ; Moved Temporarily
| "303" ; Method
| "304" ; Not Modified
| "400" ; Bad Request
| "401" ; Unauthorized
| "402" ; Payment Required
| "403" ; Forbidden
| "404" ; Not Found
| "405" ; Method Not Allowed
| "406" ; None Acceptable
| "407" ; Proxy Authentication Required
| "408" ; Request Timeout
| "409" ; Conflict
| "410" ; Gone
| "500" ; Internal Server Error
| "501" ; Not Implemented
| "502" ; Bad Gateway
| "503" ; Service Unavailable
| "504" ; Gateway Timeout
| extension-code

extension-code = 3DIGIT

Reason-Phrase = token *(SP token)

3.2 Response Header Fields

Response-Header = Public
| Retry-After
| Server
| WWW-Authenticate
| extension-header

Public = "Public" ":" 1#Method

Retry-After = "Retry-After" ":" (HTTP-date | delta-seconds

Server = "Server" ":" 1*(product)

WWW-Authenticate = "WWW-Authenticate" ":" challenge

4. Entity

4.1 Entity Header Fields

Entity-Header = Allow
| Content-Encoding
| Content-Language
| Content-Length
| Content-Transfer-Encoding
| Content-Type
| Derived-From
| Expires
| Last-Modified
| Link
| Location
| Title
| URI-header
| Version
| extension-header

IPR2025-00714
DATABRICKS EX1008 Page 304

```
extension-header  = HTTP-header

Allow             = "Allow" ":" 1#Method

Content-Encoding  = "Content-Encoding" ":"
                    encoding-mechanism

Content-Language  = "Content-Language" ":"
                    1#language-tag

Content-Length    = "Content-Length" ":"
                    1*DIGIT

Content-Transfer-
Encoding          = "Content-Transfer-Encoding"
                    ":" transfer-ding

Content-Type      = "Content-Type" ":" media-
                    type

Derived-From      = "Derived-From" ":" version-
                    tag

Expires           = "Expires" ":" HTTP-date

Last-Modified     = "Last-Modified" ":" HTTP-
                    date

Link              = "Link" ":" 1#("<" URI ">"
                    [ ";" "rel" "=" relation ]
                    [ ";" "rev" "=" relation ]
                    [ ";" "title" "=" quoted-
                    string ] )

relation          = sgml-name

sgml-name         = ALPHA *( ALPHA | DIGIT |
                    "." | "-" )

Location          = "Location" ":" URI

Title             = "Title" ":" *text

URI-header        = "URI" ":" 1#( "<" URI ">"
                    [ ";" vary ] )

vary              = "vary" "=" <"> 1#vary-
                    dimension <">

vary-dimension    = "type" | "language" |
                    "version" | "encoding"
                    | "charset" | "user-agent" |
                    extension-vary

extension-vary    = token

Version           = "Version" ":" version-tag

version-tag       = token | quoted-string
```

4.2 Entity Body

```
Entity-Body       = *OCTET
```

5. Protocol Parameters

5.1 HTTP Version

```
HTTP-Version      = "HTTP" "/" 1*DIGIT "."
                    1*DIGIT
```

5.2 Universal Resource Identifiers

```
URI               = <As defined in RFC 1630>
```

5.3 Date/Time Formats

```
HTTP-date         = rfc1123-date
                    | rfc850-date
                    | asctime-date

rfc1123-date      = wkday "," SP date1 SP time
                    SP "GMT"

rfc850-date       = weekday "," SP date2 SP
                    time SP "GMT"

asctime-date      = wkday SP date3 SP time SP
                    4DIGIT
```


IPR2025-00714
DATABRICKS EX1008 Page 305

date1 = 2DIGIT SP month SP 4DIGIT

date2 = 2DIGIT "-" month "-" 2DIGIT

date3 = month SP (2DIGIT | (SP 1DIGIT))

time = 2DIGIT ":" 2DIGIT ":" 2DIGIT

wkday = "Mon" | "Tue" | "Wed" | "Thu" | "Fri" | "Sat" | "Sun"

weekday = "Monday" | "Tuesday" | "Wednesday" | "Thursday" | "Friday" | "Saturday" | "Sunday"

month = "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"

delta-seconds = 1*DIGIT

6. Content Parameters

6.1 Media Types

media-type = type "/" subtype *(";" parameter)

type = token

subtype = token

parameter = attribute "=" value

attribute = token

value = token | quoted-string

6.2 Character Sets

charset = "US-ASCII" | "ISO-8859-1" | "ISO-8859-2" | "ISO-8859-3" | "ISO-8859-4" | "ISO-8859-5" | "ISO-8859-6" | "ISO-8859-7" | "ISO-8859-8" | "ISO-8859-9" | "ISO-2022-JP" | "ISO-2022-JP-2" | "ISO-2022-KR" | "UNICODE-1-1" | "UNICODE-1-1-UTF-7" | "UNICODE-1-1-UTF-8" | token

6.3 Encoding Mechanisms

encoding-mechanism = "gzip" | "compress" | token

6.4 Transfer Encodings

transfer-encoding = "binary" | "8bit" | "7bit" | "quoted-printable" | "base64" | token

6.5 Language Tags

language-tag = primary-tag *("-" subtag)

primary-tag = 1*8ALPHA

subtag = 1*8ALPHA

7. Access Authentication

auth-scheme = "Basic" | token

auth-param = token "=" quoted-string

challenge = auth-scheme 1*LWS realm ["," 1#auth-param]

IPR2025-00714
DATABRICKS EX1008 Page 306

realm = "Realm" "=" quoted-string

credentials = auth-scheme [1*LWS encoded-cookie] #auth-param

encoded-cookie = 1*<any CHAR except CTLs or tspecials, but including "=" and "/">

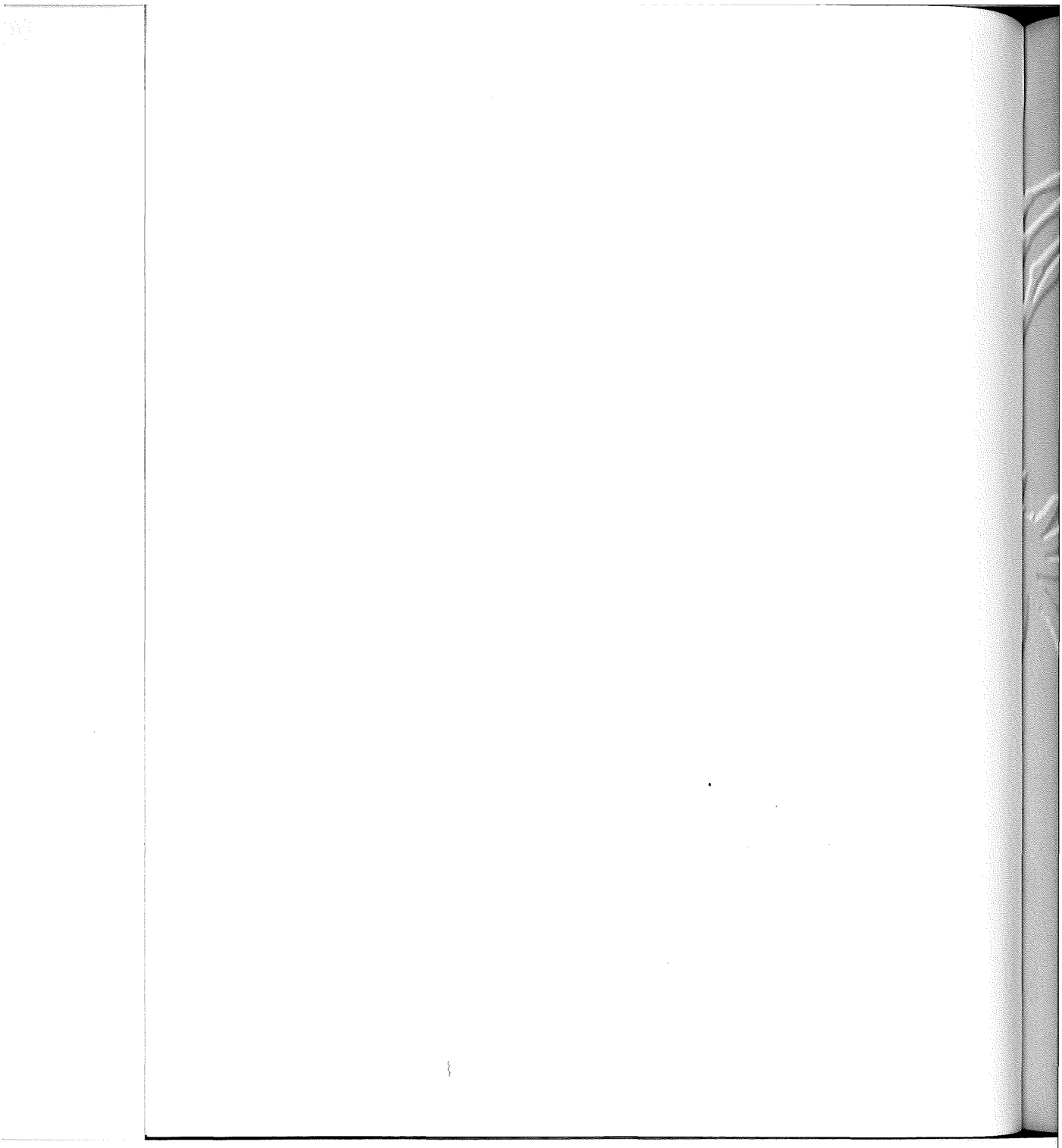
basic-challenge = "Basic" SP realm

basic-credentials = "Basic" SP basic-cookie

basic-cookie = <base64 encoding of userid-password>

userid-password = [token] ":" *text

IPR2025-00714
DATABRICKS EX1008 Page 307



appendix B

WebWalker 1.00 Program Listing

This appendix is in its entirety the WebWalker 1.00 spider program, written by the author of this book. It includes text and comments throughout the program to enhance and explain the program code.

WebWalker is based upon code and architecture derived from Roy Fielding's publicly available MOMspider progrram. Fielding's MOMspider, for Multi-Owner Maintenance spider, traverses the

IPR2025-00714
DATABRICKS EX1008 Page 309

World Wide Web to perform Web maintenance. For teaching purposes, a simpler World Wide Web robot called WebWalker was built to illustrate the important characteristics and internal architecture of a Web spider that performs Web maintenance. WebWalker performs World Wide Web traversal for individual sites and tests for the integrity of all hyperlinks to external sites.

```
#!/usr/local/bin/perl

# ==============================================================================
# Copyright (c) 1995 Fah-Chun Cheong. All rights reserved.
# ------------------------------------------------------------------------------
# WebWalker 1.00 -- A World Wide Web spider for diagnosis and maintenance of
#                   distributed hypertext infostructures.
# ==============================================================================
#
# WebWalker is based upon code and architecture derived from Roy Fielding's
# publicly available MOMspider program. Roy Fielding's MOMspider, for
# Multi-Owner Maintenance spider, traverses the World Wide Web to perform
# Web maintenance. For teaching purposes, we have built a simpler WWW robot
# called WebWalker to illustrate the important characteristics and internal
# architecture of a WWW spider that performs Web maintenance. WebWalker
# performs WWW traversal for individual sites and tests for the integrity
# of all hyperlinks to external sites.
#
# ==============================================================================
# Before changing things here, the installer should first set three things:
#   (1) The first line which specifies the perl interpreter;
#   (2) The @INClude path for libwww-perl library packages $WWWlib;
#   (3) The $LocalNetwork definition using network domain name.
# ==============================================================================

$LibWWW  = "/usr/local/lib/libwww-perl-0.40";
$WWWlib  = ($ENV{'LIBWWW_PERL'} || $LibWWW || '.');
unshift(@INC, $WWWlib);

$Version = 'WebWalker/1.00';
$HOMEdir = ($ENV{'HOME'} || $ENV{'home'} || '.');
$PWDdir  = ($ENV{'PWD'}  || $ENV{'cwd'}  || '.');
$TMPdir  = ($ENV{'TMPDIR'}               || '/tmp');

require "getopts.pl";                    # Perl library package
require "www.pl";                        # libwww-perl library package
require "wwwurl.pl";                     # ditto
require "wwwdates.pl";                   # ditto
```


IPR2025-00714
DATABRICKS EX1008 Page 310

```
require "wwwhtml.pl";                    # ditto
require "wwwmime.pl";                    # ditto

# ============================================================================
# Local Network should be the network domain which you consider to be local.
# (i.e., a network request to sites in this domain does not create any external
# network costs to your organization).  YOU WILL WANT TO CHANGE THIS!!!
# ============================================================================

$LocalNetwork = '\.agent\.com';          # Use backslash to escape periods

# ============================================================================
# Set the default location of the task instruction file of allowed tasks.
# ============================================================================

$TaskFile = "$HOMEdir/.webwalk";

# ============================================================================
# The following options allow WebWalker to decode traversable response
# content that has been encoded (so far, this only means compressed).
# This may never be used if your site does not compress any HTML files.
# ============================================================================

# The following association maps content-encodings to their decoder command.

%CEdecoder = (
    'x-compress', 'uncompress',          # Decode Adaptive Lempel-Ziv
    'x-gzip',     'gunzip',              # Decode LZ77
);

# The following association maps content-encodings to the file extension
# expected by the decoder.

%CEextension = (
    'x-compress', '.Z',                  # Adaptive Lempel-Ziv encoding
    'x-gzip',     '.gz',                 # LZ77 encoding
);

# The following sets the temporary filename [without .(Z¦gz) suffix] for
# file decoding.

$CEfile = "$TMPdir/webdoc$$-comp.html";
```


IPR2025-00714
DATABRICKS EX1008 Page 311

```
# ==============================================================================
# The following is the standard name for the URL that defines for any site
# where robots are not allowed. See Martijn Koster's proposal at:
# http://web.nexor.co.uk/mak/doc/robots/norobots.html for more info.
# ==============================================================================

$RobotsURL   = "/robots.txt";

# ==============================================================================
# Set things which control the traversal process.
# ==============================================================================

$BaseURL     = "file://localhost$PWDdir/";
                        # The initial base URL to use if TopURL is relative

$MaxDepth    = 20;      # Default maximum traversal depth. Can be overridden
                        #   by task instructions or command line.

$Timeout     = 30;      # Maximum number of seconds to wait for a response.
                        # Increase if you have a slow network connection.

$MaxConsec   = 50; #15; # Maximum number of consecutive requests to any site
                        #   before a long pause is required.

$PauseTime   = 1; #20;  # The number of seconds for a long pause.
                        # Increase if your server is very slow.

$BetweenTime = 0; #5;   # Amount of time required between any two requests
                        #   to the same site. Increase if server is slow.

# ==============================================================================
# Define symbolic names for HTTP response codes.
# ==============================================================================

$RC_unknown              = 000;
$RC_ok                   = 200;
$RC_created              = 201;
$RC_accepted             = 202;
$RC_partial              = 203;
$RC_no_response          = 204;
$RC_moved                = 301;
$RC_found                = 302;
$RC_method               = 303;
```


IPR2025-00714
DATABRICKS EX1008 Page 312

```
$RC_not_modified          = 304;
$RC_bad_request           = 400;
$RC_unauthorized          = 401;
$RC_payment_required      = 402;
$RC_forbidden             = 403;
$RC_not_found             = 404;
$RC_internal_error        = 500;
$RC_not_implemented       = 501;
$RC_bad_response          = 502;
$RC_too_busy              = 503;
$RC_bad_request_client    = 600;
$RC_not_implemented_client = 601;
$RC_connection_failed     = 602;
$RC_timed_out             = 603;

# ========================================================================
# Define all HTTP response messages indexed by response code.
# ========================================================================

%RespMessage = (
    000, 'Unknown Error',
    200, 'OK',
    201, 'CREATED',
    202, 'Accepted',
    203, 'Partial Information',
    204, 'No Response',
    301, 'Moved',
    302, 'Found',
    303, 'Method',
    304, 'Not Modified',
    400, 'Bad Request',
    401, 'Unauthorized',
    402, 'Payment Required',
    403, 'Forbidden',
    404, 'Not Found',
    500, 'Internal Error',
    501, 'Not Implemented',
    502, 'Bad Response',
    503, 'Too Busy',
    600, 'Bad Request in Client',
    601, 'Not Implemented in Client',
    602, 'Connection Failed',
    603, 'Timed Out',
);
```


IPR2025-00714
DATABRICKS EX1008 Page 313

```
# ====================================================================
# The set of all possible actions after issuing 'GET' or 'HEAD' requests.
# ====================================================================

$DO_continue = 1;                           # Continue processing
$DO_ok_stop  = 2;                           # Stop processing, no error
$DO_redirect = 3;                           # Redirected URL
$DO_broken   = 4;                           # Signal broken link

# ====================================================================
# Figure out what to do given the following response codes. Note that many
# of these should never be generated by a normal 'GET' or 'HEAD' request.
# ====================================================================

%WhatToDo = (
    $RC_unknown,               $DO_broken,     # 000
    $RC_ok,                    $DO_continue,   # 200
    $RC_created,               $DO_ok_stop,    # 201
    $RC_accepted,              $DO_ok_stop,    # 202
    $RC_partial,               $DO_continue,   # 203
    $RC_no_response,           $DO_ok_stop,    # 204
    $RC_moved,                 $DO_redirect,   # 301
    $RC_found,                 $DO_redirect,   # 302
    $RC_method,                $DO_ok_stop,    # 303
    $RC_not_modified,          $DO_ok_stop,    # 304
    $RC_bad_request,           $DO_broken,     # 400
    $RC_unauthorized,          $DO_ok_stop,    # 401
    $RC_payment_required,      $DO_ok_stop,    # 402
    $RC_forbidden,             $DO_broken,     # 403
    $RC_not_found,             $DO_broken,     # 404
    $RC_internal_error,        $DO_broken,     # 500
    $RC_not_implemented,       $DO_broken,     # 501
    $RC_bad_response,          $DO_broken,     # 502
    $RC_too_busy,              $DO_broken,     # 503
    $RC_bad_request_client,    $DO_broken,     # 600
    $RC_not_implemented_client, $DO_ok_stop,   # 601
    $RC_connection_failed,     $DO_broken,     # 602
    $RC_timed_out,             $DO_broken,     # 603
);
```


IPR2025-00714
DATABRICKS EX1008 Page 314

```
# ==========================================================================
# Status of a seen node (yes, the order is very important).
# ==========================================================================

$S_not_seen        = 0; # Node has not yet been seen (also undefined)
$S_seen_not_tested = 1; # Node has been seen but not yet tested
$S_avoided         = 2; # Node has been avoided
$S_will_leaf       = 3; # Node will be  leafed
$S_will_test       = 4; # Node will be  tested
$S_tested_unknown  = 5; # Node has been tested but has unknown leaf status
$S_leafed          = 6; # Node has been tested and determined to be a leaf
$S_will_traverse   = 7; # Node has been placed on the traversal queue
$S_traversed       = 8; # Node has been traversed

# ==========================================================================
# Initialize the infostructure task tables.
# ==========================================================================

@TaskName         = (); # Required task name for descriptive use
@TaskTopURL       = (); # Required WWW URL for the starting top document
@TaskBoundURL     = (); # Required task boundary URL prefix
@TaskChangeWindow = (); # Optional days a change is still interesting
@TaskExpireWindow = (); # Optional days before expiring is interesting
@TaskExclude      = (); # Optional URLs to exclude (leaf) from this task

# ==========================================================================
# Initialize the sites, avoids and leaf tables.
# ==========================================================================

$SitesNum    = 0;       # Number of entries in @SitesAddr
@SitesAddr   = ();      # Sites table
%Sites       = ();      # Reverse sites table (for duplication detection)

$AvoidNum    = 0;       # Number of entries in @AvoidURL
@AvoidURL    = ();      # Avoids table

$LeafNum     = 0;       # Number of entries in @LeafURL
@LeafURL     = ();      # Leaf table

# ==========================================================================
# Initialize traversal history information.
# ==========================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 315

```
$VisNumber   = 0;       # Number of URLs visited since process start
%Visited     = ();      # Associative array of URLs visited -> @Vis* index
@VisURL      = ();      # URL of node (maps @Vis* index -> URL visited)
@VisStatus   = ();      # Status of a seen node
@VisRespCode = ();      # Server response code from last access

@VisConType  = ();      # MIME Content-type of response
@VisRedirect = ();      # Redirected URL (from a 302 Moved response)
@VisTitle    = ();      # Title text from headers or last traversal
@VisOwner    = ();      # Owner name from headers or last traversal
@VisReplyTo  = ();      # Reply-To address from headers or last traversal
@VisLastMod  = ();      # Last-modified date from headers
@VisExpires  = ();      # Expires date from headers

@VisInTask   = ();      # Seen during the current task?
@VisLocal    = ();      # URL considered to be local to this network?

# ============================================================================
# Initialize nodes traversal data structures.
# ============================================================================

$CurConsec   =  1;     # Current number of consecutive requests to a site
$PrevSite    = '';     # Site of the last network request
$PrevTime    =  0;     # Time of the last network request

@TravNodes   = ();     # Nodes that we have yet to traverse for this task
@TravDepth   = ();     #   and their traversal depth
@TravParent  = ();     #   and their parent's url

# ============================================================================
# The statistical table summary format generally looks like this:
#
#            .-------------------------------------------.
#            |  References   |  Unique URLs  |  Local URLs   |
#            | number   pct  | number   pct  | number   pct  |
#            |---------------+---------------+---------------|
# Traversed  | NNNNNN NNN.NN | NNNNNN NNN.NN | NNNNNN NNN.NN |
# Tested     | NNNNNN NNN.NN | NNNNNN NNN.NN | NNNNNN NNN.NN |
# Reused     | NNNNNN NNN.NN | NNNNNN NNN.NN | NNNNNN NNN.NN |
# Avoided    | NNNNNN NNN.NN | NNNNNN NNN.NN | NNNNNN NNN.NN |
# Untestable | NNNNNN NNN.NN | NNNNNN NNN.NN | NNNNNN NNN.NN |
#            |---------------+---------------+---------------|
# Broken     | NNNNNN NNN.NN | NNNNNN NNN.NN | NNNNNN NNN.NN |
```


IPR2025-00714
DATABRICKS EX1008 Page 316

```
# Redirected ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦
# Changed NN ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦
# Expired NN ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦
#            ¦--------------+--------------+--------------¦
# Local      ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦ NNNNNN 100.00 ¦
# Remote     ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦      0   0.00 ¦
#            ¦--------------+--------------+--------------¦
# Totals     ¦ NNNNNN 100.00 ¦ NNNNNN NNN.NN ¦ NNNNNN NNN.NN ¦
#            '-----------------------------------------------'
#
# ============================================================================

$SummaryFormat = <<'EOF';
           .-----------------------------------------------.
           ¦  References   ¦  Unique URLs  ¦  Local URLs   ¦
           ¦ number   pct  ¦ number   pct  ¦ number   pct  ¦
           ¦--------------+--------------+--------------¦
Traversed  ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Tested     ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Reused     ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Avoided    ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Untestable ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
           ¦--------------+--------------+--------------¦
Broken     ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Redirected ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Changed%3d ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Expired%3d ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
           ¦--------------+--------------+--------------¦
Local      ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
Remote     ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
           ¦--------------+--------------+--------------¦
Totals     ¦%7d %6.2f      ¦%7d %6.2f      ¦%7d %6.2f      ¦
           '-----------------------------------------------'
EOF

# ============================================================================
# Get the command-line options.
# ============================================================================

if (!(&Getopts('hi:d:')) ¦¦ $opt_h) { &usage; }

if ($opt_i) { $TaskFile = $opt_i; }         # Overrides default task file
if ($opt_d) { $MaxDepth = $opt_d; }         # Overrides default max depth
```


IPR2025-00714
DATABRICKS EX1008 Page 317

```
select((select(STDERR), $| = 1)[0]);        # Make STDERR unbuffered
$| = 1;                                     # Make STDOUT unbuffered

# ========================================================================
# Main processing loop cycling over infostructure-based tasks.
# ========================================================================

print $Version, ' starting at ', &wwwdates'wtime(time, ''), "\n";
&read_task || &read_tasks;                  # From query string or tasks file

&www'set_def_header('http', 'User-Agent', $Version);

foreach $task (1 .. $#TaskName)             # For each specified task
{
    next unless $TaskName[$task];           # Skip if task not properly defined

    $LeafNum = 0;                           # Zero entries in leaf table
    @LeafURL = ();                          # Empty leaf table

    foreach $leafurl (split(/#/, $TaskExclude[$task]))
    {
        &add_leaf($leafurl);                # Add URL to exclude in leaf table
    }
                                            # Traverse the Web infostructure
    &traverse_web($task, $TaskName[$task],
                         $TaskTopURL[$task],
                         $TaskBoundURL[$task]);
}

print $Version, ' finished at ', &wwwdates'wtime(time, ''), "\n";

# ========================================================================
# Print usage information if help requested (-h) or an invalid option given.
# ========================================================================
sub usage
{
    die <<"EndUsage";
usage: webwalk [-h] [-i taskfile] [-d maxdepth]
$Version
WWW Robot for maintenance of distributed hypertext infostructures.
Options:                                                    [DEFAULT]
     -h  Help — just display this message and quit.
     -i  Get your task instructions from the following file.  [$TaskFile]
```


IPR2025-00714
DATABRICKS EX1008 Page 318

```
    -d  Maximum traversal depth.                          [$MaxDepth]
EndUsage
}

# ========================================================================
# Handle GET and POST method parameters if used as CGI script.
# ========================================================================
sub read_task
{
    local($qstr);                          # Query string

    if ($ENV{'REQUEST_METHOD'} eq "GET")   # GET method?
    {
        $qstr = $ENV{'QUERY_STRING'};      # Extract query string
    }
    elsif ($ENV{'REQUEST_METHOD'} eq "POST")# POST method?
    {
        read(STDIN, $qstr, $ENV{'CONTENT_LENGTH'});
    }
    else { return $CGI = 0; }              # Script not used in CGI context

    $qstr =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C",hex($1))/ge;
    $qstr =~ s/\+/ /g;                     # Restore blanks in argument values
    local(%env) = split (/[&=]/, $qstr);   # Parse into associative array
    local(@env) = split (/&/, $qstr);      # Parse into list

    $TaskName[1]         = $env{'Name'};
    $TaskTopURL[1]       = $env{'TopURL'};
    $TaskBoundURL[1]     = $env{'BoundURL'};
    $TaskChangeWindow[1] = $env{'ChangeWindow'};
    $TaskExpireWindow[1] = $env{'ExpireWindow'};
    $TaskExclude[1]      = $env{'Exclude'};
    $MaxDepth            = $env{'MaxDepth'};
    $ReplyTo             = $env{'ReplyTo'};

    if ($ReplyTo)                          # Email address of WebWalker user
    {
        &www'set_def_header('http', 'From', $ReplyTo);
    }

    print "Content-type: text/html\012\012";

    if    (!$TaskName[1])     { die "Task has no Name";     }
```


IPR2025-00714
DATABRICKS EX1008 Page 319

```
    elsif (!$TaskTopURL[1])   { die "Task has no TopURL";   }
    elsif (!$TaskBoundURL[1]) { die "Task has no BoundURL"; }

    return $CGI = 1;                        # Script is used in CGI context
}

# ==========================================================================
# Read task instructions from the $TaskFile and fill in the @Task* tables.
# ==========================================================================
sub read_tasks
{
    local($task)   = 0;                     # Task index into @Task* tables
    local($intask) = 0;                     # Inside of task description?
    local($taskno) = 0;                     # No. of task instructions seen
    local($reason) = '';                    # Why task description is bad

    print "Reading task specifications from $TaskFile\n\n";

    if (!open(TASK, $TaskFile))             # Open task file for read
    {
        print STDERR "Cannot open task file: $!\n";
        &usage;                             # Show proper usage
    }

    while (<TASK>)                          # For each line of task file
    {
        next if (/^$/ ¦¦ /^\#/);            # Ignore blank and comment lines

        if (!$intask)                       # If not within a task?
        {
            $taskno++;                      # Increment task instruction no.
            if (/^</)                       # Start of next task?
            {                               # Fill in the defaults if needed
                $task++;                    # New task index
                $TaskChangeWindow[$task] = 7;
                $TaskExpireWindow[$task] = 0;
                $TaskExclude[$task]      = '';
                $intask = 1;                # Now inside a task
            }
            elsif (/^MaxDepth\s+(\d+)\s/)   # Maximum depth of traversal
            {
                if (!$opt_d) { $MaxDepth = $1; }
            }
```


IPR2025-00714
DATABRICKS EX1008 Page 320

```
    elsif (/^ReplyTo\s+(\S.*)/)     # Email address of WebWalker user
    {
        &www'set_def_header('http', 'From', $1);
    }
    else
    {
        print STDERR "Unrecognized instruction at line $.\n";
        print STDERR "  of $TaskFile\n";
    }
    next;                           # Next instructions
}
                                    # Currently within a task
if (/^</)                           # Indicates beginning of task?
{
    die "Task $taskno is not properly terminated, stopped";
}
elsif (/^\s*Name\s+((\S+\s*)+)\n$/) { $TaskName        [$task] = $1; }
elsif (/^\s*TopURL\s+(\S+)/)        { $TaskTopURL      [$task] = $1; }
elsif (/^\s*BoundURL\s+(\S+)/)      { $TaskBoundURL    [$task] = $1; }
elsif (/^\s*ChangeWindow\s+(\d+)/)  { $TaskChangeWindow[$task] = $1; }
elsif (/^\s*ExpireWindow\s+(\d+)/)  { $TaskExpireWindow[$task] = $1; }
elsif (/^\s*Exclude\s+([^#\s]+)/)   { $TaskExclude     [$task].= $1.'#'; }
elsif (/^>\s*$/)                    # Line indicates End of Task?
{
    $intask = 0;                    # No longer inside task
    $reason = '';                   # No bad task reason until told

    if    (!$TaskName[$task])     { $reason = "has no Name";     }
    elsif (!$TaskTopURL[$task])   { $reason = "has no TopURL";   }
    elsif (!$TaskBoundURL[$task]) { $reason = "has no BoundURL"; }

    if ($reason)                    # If task requirement was not met
    {                               #   then undo its task options
        print(STDERR "Task $taskno ", $reason, ", skipping it.\n");
        undef $TaskName        [$task];
        undef $TaskTopURL      [$task];
        undef $TaskBoundURL    [$task];
        undef $TaskChangeWindow[$task];
        undef $TaskExpireWindow[$task];
        undef $TaskExclude     [$task];
        $task--;                    # Bad task don't count
    }
}
```


IPR2025-00714
DATABRICKS EX1008 Page 321

```
        else
        {
            print STDERR "Unrecognized option in task $taskno\n";
            print STDERR "  at line $. of $TaskFile\n";
        }
    }

    if ($intask)                            # Inside task but end of file?
    {
        die "Last instruction is not properly terminated, stopped";
    }
    close TASK;                             # Close task file
}

# ============================================================================
# Check the given $url for any restrictions on its access.
# Return  0 => no restrictions on access
#         1 => leaf (okay to test, but don't traverse)
#         2 => avoid (no access allowed)
# ============================================================================
sub check_url
{
    local($url) = @_;                       # URL to be checked
    local($site);                           # URL site
    local($idx);                            # Index into Avoid and Leaf arrays
    local($prefix);                         # URL prefix

    if (($url =~ /^http:/) &&               # URL scheme is http
        ($site = &wwwurl'get_site($url)))   #   and site is specified
    {
        if (&check_site($site))             # If site not previously checked
        {
            &add_site($site);               # It is checked now
        }                                   #  and added to sites table
    }

    for ($idx = 1; $idx <= $AvoidNum; $idx++)
    {                                       # For each entry in avoid table
        next unless ($AvoidURL[$idx]);      # Skip over undefined avoid url
        $prefix = $AvoidURL[$idx];          # Prefix is URL to be avoided
        return 2 if ($url =~ m#^$prefix#);  # URL less general than prefix
    }
```


IPR2025-00714
DATABRICKS EX1008 Page 322

```
    for ($idx = 1; $idx <= $LeafNum; $idx++)
    {                                       # For each entry in sites table
        $prefix = $LeafURL[$idx];           # Prefix is URL to be leafed
        return 1 if ($url =~ m#^$prefix#);  # URL less general than prefix
    }

    return 0;                               # No restrictions on access
}

# ========================================================================
# Check the sites table to see if this $site has already been checked for
# restrictions. If it hasn't, perform the check on that site using the
# RobotsNotWanted protocol and update the avoids and sites tables. See the
# details at http://web.nexor.co.uk/mak/doc/robots/norobots.html.
# Return  0 => RobotsURL at site previously checked
#         1 => this site never checked before
# ========================================================================
sub check_site
{
    local($site) = @_;                      # Site to be checked
    local($drs);                            # List of disallowed names

    return 0 if defined($Sites{$site});     # Has RobotsURL been checked?

    local($url) = "http://$site$RobotsURL"; # RobotsNotWanted URL to check
    local(%headers) = ();                   # To hold parsed response headers
    local($headers) = '';                   # To hold response headers
    local($content) = '';                   # To hold response content

    print "Checking for $url ... ";         # Make 'GET' request on RobotsURL
    local($response) = &www'request('GET', $url, *headers, *content, $Timeout);
    print "$response $RespMessage{$response}\n";

    return 1 unless ($response == $RC_ok);  # Do not proceed if response is bad

    local($whoami)  = $Version;             # "WebWalker/1.xx"
    $whoami         =~ s#/.*##;             # Remove version or library info
    local($in_def)  = 0;                    # Default '*' record boundaries
    local($in_mine) = 0;                    # My own record boundaries
    local($def_dr)  = '';                   # Store default disallow names
    local($mine_dr) = '';                   # Store my own disallow names

    foreach (split(/\n/, $content))         # For each line in body content
```


IPR2025-00714
DATABRICKS EX1008 Page 323

```
    {
        next if (/^\s*#/);                  # Ignore comment-only lines
        s/\s*#.*//;                         # Remove any other comments

        if (/^\s*$/)                        # Records separated by blank lines
        {
            last if ($in_mine);             # Break out loop if end of mine
            $in_def = 0;                    # Not in default '*' by default
        }
        elsif (/^\s*User-Agent:\s*(.*)\s*$/i)
        {                                   # List of robot names
            next if ($in_mine);
            local($agent) = $1;
            if ($agent =~ /\b$whoami\b/i) { $in_mine = 1; next; }
            if ($agent =~ /^\*/)          { $in_def  = 1; }
        }
        elsif (/^\s*Disallow:\s*(.*)\s*$/i)
        {                                   # List of URLs to avoid
            next unless ($in_def || $in_mine);
            $drs = $1;
            if ($in_mine) { $mine_dr .= ' '. $drs; next; }
            if ($in_def)  { $def_dr  .= ' '. $drs; }
        }
    }

    if ($in_mine) { $def_dr = $mine_dr; }   # My own record takes precedence

    if ($def_dr !~ /^\s*$/)                 # If not a blank line
    {
        foreach $drs (split(' ', $def_dr))  # For each URL to avoid
        {                                   # Add to avoids table
            &add_avoid(&wwwurl'absolute($url, $drs));
        }
    }
    return 1;                               # New site is now checked
}

# ==========================================================================
# Add the given $site to the sites table, duplication will be detected.
# ==========================================================================
sub add_site
{
    local($site) = @_;                      # Site to be added to table
```


IPR2025-00714
DATABRICKS EX1008 Page 324

```
    local($idx)  = $Sites{$site};          # Retrieve index based upon site

    if (!$idx)                             # Index not defined?
    {
        $idx = ++$SitesNum;                # Must create new index
        $Sites{$site} = $idx;              # Make room for new site
    }
    $SitesAddr[$idx] = $site;              # Enter into sites table
}

# ========================================================================
# Add the given $url to the avoids table. All existing avoid or leaf entries
# will be checked first for duplication or overlapping.
# ========================================================================
sub add_avoid
{
    local($url) = @_;                      # URL to be added to avoids table
    local($old);                           # Old URL in avoids table
    local($idx);                           # Index into avoids table
    local($pos);                           # Position in avoids table

    undef $pos;                            # Set to be undefined

    foreach $idx (1 .. $AvoidNum)          # For each node in avoids table
    {
        if (!$AvoidURL[$idx])              # Gap exists in avoids table?
        {
            $pos = $idx;  next;            # Fill any gaps
        }
        $old = $AvoidURL[$idx];            # Recall old URL to be matched
        if (($url eq $old) ||              # URL duplicates an old one?
            ($url =~ m#^$old#))            # URL less general than old one?
        {
            return;                        # Don't need to add to table
        }
        if ($old =~ m#^$url#)              # URL more general than old one?
        {
            $pos = $idx;  last;            # Update with more general URL
        }
    }

    if (!defined($pos)) { $pos = ++$AvoidNum; }
    $AvoidURL[$pos] = $url;                # Add URL to avoids table
}
```


IPR2025-00714
DATABRICKS EX1008 Page 325

```
# ============================================================================
# Add the given $url to the leaf table for the duration of the current
# traversal (useful for delineating the bounds of a traversal process).
# All existing leaf entries will be checked for duplication or overlapping.
# ============================================================================
sub add_leaf
{
    local($url) = @_;                   # URL to be added to leaf table
    local($old);                        # Old URL in leaf table
    local($idx);                        # Index into leaf table

    foreach $idx (1 .. $LeafNum)        # For each node in leaf table
    {
        $old = $LeafURL[$idx];          # Recall old URL to be matched
        return if ($url =~ m#^$old#);   # URL less general than old one?
    }
    $LeafURL[++$LeafNum] = $url;        # Add URL to leaf table
}

# ============================================================================
# Set the current traversal $status of the given $node, provided it is more
# advanced than previously recorded status.
# Return true (1) if okay, else false (0) if $node not found.
# ============================================================================
sub set_status
{
    local($node, $status) = @_;         # Status of node to be set

    return 0 unless defined($VisURL[$node]);# Node not found in history

    local($cstat) = $VisStatus[$node];  # Recall node traversal status

    if (!defined($cstat) ||             # Undefined old status or
        ($cstat < $status))             # Current status more advanced
    {
        $VisStatus[$node] = $status;    # Set current improved status
    }
    return 1;                           # Status ok
}

# ============================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 326

```
# Reset the status of all visited nodes in history so that they are no
# longer considered traversed.
# ===========================================================================
sub reset_status
{
    local($node);

    foreach $node (1 .. $VisNumber)         # For each node in history
    {
        next unless defined($VisURL[$node]);# Skip over holes in history
        $VisInTask[$node] = 0;              # Not considered in this task

        if ($VisStatus[$node] > $S_tested_unknown)
        {
            $VisStatus[$node] = $S_tested_unknown;
        }
        elsif (($VisStatus[$node] == $S_seen_not_tested) ||
               ($VisStatus[$node] == $S_will_leaf)       ||
               ($VisStatus[$node] == $S_will_test))
        {
            $VisStatus[$node] = $S_not_seen;
        }
    }
}

# ===========================================================================
# Check to see if the passed-in absolute $url is in history. If so, update
# its $status if the new status is more advanced and return the $node index
# to its history record. If not, create a history record for the new $url
# with the given $status.
# ===========================================================================
sub remember
{
    local($url, $status) = @_;              # URL and status to be remembered
    local($node) = $Visited{$url};          # Retrieve node index from URL

    if (!$node)                             # If node not found in history
    {
        $node              = ++$VisNumber; # Create new node index
        $Visited{$url}     = $node;        # Remember this node
        $VisURL[$node]     = $url;         # Remember this URL
        $VisStatus[$node]  = $status;      # Remember URL's status
        $VisRespCode[$node] = 0;           # Init http response code
```


IPR2025-00714
DATABRICKS EX1008 Page 327

```
        if (($url =~ m#^file:#) ||          # Is URL local?
            ($url =~ m#://[^/]*$LocalNetwork#io))
        {
            $VisLocal[$node] = 1;           # Yes, mark it as local
        }
    }
    else                                    # Node is found in past history
    {
        &set_status($node, $status);        # Update status of node in history
    }

    $VisInTask[$node] = 1;                  # This node is in current task
    return $node;                           # Return node index
}

# ========================================================================
# Store and update history from meta information held in *headers for the
# given $node along with its $status and $response code from a recent WWW
# request. Return 1 if okay, or 0 if the $node was not found in history.
# ========================================================================
sub store
{
    local($node, $status, $response, *headers) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?

    if (($VisStatus[$node] == $S_will_leaf) &&          # Will be leafed and
        ($status           == $S_tested_unknown))       #   already tested?
    {
        $status = $S_leafed;                            # Then treat as leafed
    }
    &set_status($node, $status);                        # Store status

    $VisConType[$node]  =  $headers{'content-type'};    # MIME Content-type
    $VisRedirect[$node] = ($headers{'uri'} ||
                           $headers{'location'});       # Redirected URL
    $VisTitle[$node]    =  $headers{'title'};           # Title text
    $VisOwner[$node]    =  $headers{'owner'};           # Owner name
    $VisReplyTo[$node]  =  $headers{'reply-to'};        # Reply-To address
    $VisLastMod[$node]  =  $headers{'last-modified'};   # Last-modified date
    $VisExpires[$node]  =  $headers{'expires'};         # Expires date
    $VisRespCode[$node] =  $response;                   # Latest response code
```


IPR2025-00714
DATABRICKS EX1008 Page 328

```
    return 1;                                       # Store operation ok
}

# ============================================================================
# Recall meta information stored in history for the given $node, to be
# passed back inside the %headers array. Return the most recent response
# code or 0 if the given $node was not found in history.
# ============================================================================
sub recall
{
    local($node, *headers) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?

    $headers{'content-type'}  = $VisConType[$node];     # MIME Content-type
    $headers{'uri'}           = $VisRedirect[$node];    # Redirected URL
    $headers{'title'}         = $VisTitle[$node];       # Title text
    $headers{'owner'}         = $VisOwner[$node];       # Owner name
    $headers{'reply-to'}      = $VisReplyTo[$node];     # Reply-To address
    $headers{'last-modified'} = $VisLastMod[$node];     # Last-modified date
    $headers{'expires'}       = $VisExpires[$node];     # Expires date

    return $VisRespCode[$node];                         # Latest response code
}

# ============================================================================
# Return true (1) if the given $node has been avoided.
# Return false (0) if the given $node was not found in history.
# ============================================================================
sub was_avoided
{
    local($node) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?
    return ($VisStatus[$node] == $S_avoided);
}

# ============================================================================
# Return true (1) if the given $node has been tested.
# Return false (0) if the given $node was not found in history.
# ============================================================================
sub was_tested
{
```


IPR2025-00714
DATABRICKS EX1008 Page 329

```
    local($node) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?
    return ($VisStatus[$node] >= $S_tested_unknown);
}

# ============================================================================
# Return true (1) if the given $node is untestable.
# Return false (0) if the given $node was not found in history.
# ============================================================================
sub is_untestable
{
    local($node) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?
    return ($VisRespCode[$node] == $RC_not_implemented_client);
}

# ============================================================================
# Return true (1) if the given $node will be or has already been checked for
# its traversal status for this infostructure.
# Return false (0) if the given $node was not found in history.
# ============================================================================
sub is_known
{
    local($node) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?
    return ($VisStatus[$node] > $S_tested_unknown);
}

# ============================================================================
# Return true (1) if the given $node will be or has already been traversed
# (or leafed) for this infostructure.
# Return false (0) if the given $node was not found in history.
# ============================================================================
sub is_traversing
{
    local($node) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?
    return ($VisStatus[$node] >= $S_will_traverse);
}
```


IPR2025-00714
DATABRICKS EX1008 Page 330

```
# ==========================================================================
# Return true (1) if the given $node has been traversed.
# Return false (0) if the given $node was not found in history.
# ==========================================================================
sub was_traversed
{
    local($node) = @_;

    return 0 unless defined($VisURL[$node]);            # Node not found?
    return ($VisStatus[$node] == $S_traversed);
}

# ==========================================================================
# Return true if given $node is considered local. Return undef otherwise.
# ==========================================================================
sub is_local
{
    return $VisLocal[$_[0]];
}

# ==========================================================================
# Return the stored URL of the given $node. Return undef if not found.
# ==========================================================================
sub get_url
{
    return $VisURL[$_[0]];
}

# ==========================================================================
# Traverse an entire infostructure in breadth-first manner. The
# infostructure is indexed at $task and named $taskname. Start traversal at
# $tasktop URL and continuing until there are no more nodes to traverse
# within the infostructure bounded by $taskbound, or $MaxDepth is reached.
# ==========================================================================
sub traverse_web
{
    local($task, $taskname, $tasktop, $taskbound) = @_;

    &begin_summary($task);                  # Start of infostructure traversal

    local($url)    = &wwwurl'absolute($BaseURL, $tasktop);
    local($node)   = &remember($url, $S_seen_not_tested);
```


IPR2025-00714
DATABRICKS EX1008 Page 331

```
local($depth)  = 0;                    # Current depth starts at zero
local($parent) = '';                   # Parent of current node
$tasktop       = $url;                 # The absolute form of tasktop

if (&should_avoid($node))              # Node should be avoided?
{
    print "Avoided Top URL $url ... ";
    &traversed($node);                 # Node considered traversed
    undef $node;                       # Node is undefined now
}

while ($node)                          # While more nodes in queue
{
    next if (!$url);                   # Skip in case node was deleted

    print "Traversing $url ... ";
    &traverse_link($node, $parent);    # Traverse link to GET node URL
    &traversed($node);                 # Node marked as traversed

    while ($TestLinks[0])              # While more links to test
    {
        local($child) = shift(@TestLinks);  # $url is now the parent
        local($abs)   = shift(@TestAbs);    # Absolute URL
        local($orig)  = shift(@TestOrig);   # Original href
        local($type)  = shift(@TestType);   # Anchor type

        next if (!$child);             # Skip if child not defined
                                       # Remember child as seen
        local($vidx) = &remember($child, $S_seen_not_tested);
        local($reused);                # Flag indicates reuse of test

        if (&was_tested($vidx))        # Was child tested before?
        {
            print "Reusing test of $child ... ";
            $reused = 1;               # Yes, previous result reused
        }
        else
        {
            if (&should_avoid($vidx))  # Should child be avoided?
            {
                print "Avoiding $child ... ";
                &tested($vidx, 1);     # Child considered tested
                next;                  # Next child to test
            }
```


IPR2025-00714
DATABRICKS EX1008 Page 332

```
            print "Testing $child ... ";
            &test_link($vidx, $url);    # Test url link with HEAD
            $reused = 0;                # Mark as not reused
        }

        if ($depth >= $MaxDepth)        # Maximum depth exceeded?
        {                               # Don't go beyond max depth
            ; # Do Nothing              # Don't mark as leaf either
        }
        elsif (&should_traverse($vidx, $type, $taskbound))
        {                               # Child should be traversed
            local($pos)       = $#TravNodes + 1;
            $TravNodes[$pos]  = $vidx;
            $TravDepth[$pos]  = $depth + 1;
            $TravParent[$pos] = $url;
            &set_status($vidx, $S_will_traverse);
        }
        else                            # Child should be leafed
        {
            &set_status($vidx, $S_leafed);
        }

        &tested($vidx, $reused);        # Mark child as tested
    }
    print "Done Traversing $url ...\n... at ", &wwwdates'wtime(time,''),
          "-- ", $#TravNodes + 1, " remaining on queue\n\n";
}
continue
{
    $node   = shift(@TravNodes);        # Next in queue to examine
    $depth  = shift(@TravDepth);
    $parent = shift(@TravParent);
    if ($node) { $url = &get_url($node); }
}

&end_summary($task);                    # End of infostructure traversal
&reset_status;                          # Reset status of node history
}

# ==========================================================================
# Return true (1) if the given $node should be or has already been avoided.
# Return false (0) otherwise.
# ==========================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 333

```
sub should_avoid
{
    local($node) = @_;                    # Should this node be avoided?

    # Has this node already been avoided? So should avoid.
    # Has this node already been checked? So should not avoid.

    return 1 if ( $VisStatus[$node] == $S_avoided);
    return 0 if (($VisStatus[$node] >  $S_avoided) &&
                ($VisStatus[$node] != $S_tested_unknown));

    local($url) = $VisURL[$node];         # Get this node's URL
    return 1 unless ($url);               #   or should avoid if not found

    local($check) = &check_url($url);     # Check Avoid/Leaf status
    return 0 unless ($check);             # Check 0 => no restrictions

    if ($check == 1)                      # Check 1 => must leaf this URL
    {
        $VisStatus[$node] = ($VisStatus[$node] == $S_tested_unknown)
                          ?  $S_leafed : $S_will_leaf;
        return 0;                         # Don't have to avoid
    }

    $VisStatus[$node] = $S_avoided;       # Check 2 => must avoid this URL
    return 1;                             # Should avoid this URL
}

# ========================================================================
# Determine whether or not the given $node should be traversed or leafed for
# the current infostructure. Don't traverse if:
#     (1) node $type is Image or Queries;
#     (2) node should be avoided;
#     (3) node already traversed (or will soon);
#     (4) result from previous test was not OK;
#     (5) node content is not HTML;
#     (6) node $url is not below $taskbound.
# Return  1 => should traverse this node
#         0 => should leaf     this node
# ========================================================================
sub should_traverse
{
```


IPR2025-00714
DATABRICKS EX1008 Page 334

```
    local($node, $type, $taskbound) = @_;

    return 0 if (($type eq 'I') ||          # Don't traverse links of IMGs
                 ($type eq 'Q'));           #   or Queries anchor type

    return 0 if (&should_avoid($node));     # Don't traverse if should avoid
    return 0 if (&is_known($node));         # Don't traverse again if already
                                            #   traversed (or will soon)

    local(%headers) = ();                   # Holds headers meta information
    local($response);                       # Http response code
    $response = &recall($node, *headers);   # Recall results from 'HEAD' test

                                            # Don't traverse if test not OK
    return 0 unless (($WhatToDo{$response} == $DO_continue) ||
                     ($WhatToDo{$response} == $DO_redirect));

    local($url) = &get_url($node);          # Retrieve URL from node index
                                            # Don't traverse if not HTML
    return 0 unless (&is_html($url, *headers));

    local($scheme,  $addr,  $port,  $path,  $query,  $frag)
          = &wwwurl'parse($url);            # $url -> scheme://addr:port

    local($_scheme, $_addr, $_port, $_path, $_query, $_frag)
          = &wwwurl'parse($taskbound);      # $taskbound -> scheme://addr:port

    return 0 unless (($scheme eq $_scheme) &&
                     ($addr   eq $_addr)   &&
                     ($port   eq $_port));  # Don't traverse if not equal

    $_path =~ s#/[^/]*$#/#;                 # Trim any filename off $TaskBound
    return 0 unless ($path =~ /^$_path/);   # Check if url is below $TaskBound

    return 1;                               # Otherwise should traverse!
}

# ========================================================================
# Test URL indexed by given $node via http 'HEAD' request, may slow down to
# moderate its speed as needed. Its $parent node is actually its 'Referer'
# for purpose of providing http meta information to server. Store response
# code and headers meta information in history and update its status.
# Return the http server response code.
# ========================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 335

```
sub test_link
{
    local($node, $parent) = @_;             # Current node and its parent

    local($url)       = &get_url($node);    # Retrieve stored URL from history
    local(%headers)   = ();                 # To hold parsed response headers
    local($headers)   = '';                 # To hold response headers
    local($content)   = '';                 # To hold response content

                                            # Set up the HTTP request headers
    $headers{'Accept'}  = '*/*';            # All content types are of interest
    $headers{'Referer'} = $parent;          # Parent node refers this URL

                                            # Do not make too many consecutive
    &slow_down($url);                       #   requests too fast. May sleep...

    # Issue http 'HEAD' request for $url subject to $Timeout, pass back
    # $response code, *headers meta information and *content.

    local($response) = &www'request('HEAD', $url, *headers, *content, $Timeout);

    # Store $response code and *headers meta information in history.

    &store($node, $S_tested_unknown, $response, *headers);

    return $response;                       # Return http response code
}

# ============================================================================
# Traverse URL indexed by given $node via http 'GET' request, may slow down
# to moderate its speed as needed. Its $parent node is actually its
# 'Referer' for purpose of providing http meta information to server. Add
# any links extracted from headers or content of this $node to the @Test*
# queues for further traversal. Store response code and headers meta
# information in history and update its status. Return the http server
# response code.
# ============================================================================
sub traverse_link
{
    local($node, $parent) = @_;             # Current node and its parent

    local($url)       = &get_url($node);    # Retrieve stored URL from history
```


IPR2025-00714
DATABRICKS EX1008 Page 336

```
    local(%headers)   = ();                # To hold parsed response headers
    local($headers)   = '';                # To hold response headers
    local($content)   = '';                # To hold response content

                                           # Set up the HTTP request headers
    $headers{'Accept'}  = 'text/html';     # Only "text/html" is of interest
    $headers{'Referer'} = $parent;         # Parent node refers this URL

                                           # Do not make too many consecutive
    &slow_down($url);                      #   requests too fast. May sleep...

    # Issue http 'GET' request for $url subject to $Timeout, pass back
    # $response code, *headers meta information and *content.

    local($response) = &www'request('GET', $url, *headers, *content, $Timeout);

    # Extract links from *headers and *content for use in later traversal.

    &extract_links($url, *headers, *content, $response);

    # Store $response code and *headers meta information in history.

    &store($node, $S_traversed, $response, *headers);

    return $response;                      # Return http response code
}

# ========================================================================
# Extract $url document meta information and links from *headers and
# *content, and deposit in @Test* arrays for use in further traversal. The
# $response argument determines whether redirected link is encountered.
# ========================================================================
sub extract_links
{
    local($url, *headers, *content, $response) = @_;

    @TestLinks = ();                       # Absolute URLs (w/o query or tag)
    @TestAbs   = ();                       # Absolute URLs (w/  query or tag)
    @TestOrig  = ();                       # Original href
    @TestType  = ();                       # (Link, Image, Query, Redirect)

    if (($WhatToDo{$response} == $DO_continue) &&
        (&is_html($url, *headers)))        # Only HTML contains links
```


IPR2025-00714
DATABRICKS EX1008 Page 337

```
    {
        local($encoding) = $headers{'content-encoding'};

        if (&decode($encoding, *content))   # Decode document if compressed
        {
            # Extract $url document meta information, links, and anchor types
            # from *headers and *content, and passed back for traversal use.

            &wwwhtml'extract_links($url, *headers, *content,  *TestLinks,
                                   *TestAbs, *TestOrig, *TestType);
        }
    }

    if ($WhatToDo{$response} == $DO_redirect)
    {
        local($redir);                      # Redir treated as single child

        if ($redir = $headers{'location'})
        {
            $redir =~ s/, .*//;             # Get rid of multiple
        }                                   #   Location: entries
        elsif ($redir = $headers{'uri'})
        {
            $redir =~ s/\s*;.*//;
            $redir =~ s/, .*//;             # Eliminate multiple URI: entries
        }

        if ($redir)                         # Redirected link exists?
        {
            $TestLinks[0] = $redir;         # Absolute URL to be tested
            $TestAbs[0]   = $redir;         # Original HREF in absolute form
            $TestOrig[0]  = $redir;         # Original HREF
            $TestType[0]  = 'R';            # Redirected link type
        }
    }
}

# ============================================================================
# Given the $url to test or traverse, make sure that WebWalker is not making
# too many consecutive requests and/or going too fast with respect to a
# single site. This prevents WebWalker from completely dominating the
# resources of a particular server. The remedy is to sleep for a while
# within this routine before returning.
# ============================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 338

```
sub slow_down
{
    local($url) = @_;                       # Given URL to test or traverse
    local($site);                           # Http site extracted from URL
    local($secs);                           # Wake up time (secs after Epoch)

    return unless ($url =~ m#^http#);       # Applicable to http only
    return unless ($site = &wwwurl'get_site($url));

    LOITER: {
        if ($site ne $PrevSite)             # Not same as previous site?
        {
            $CurConsec = 1;                 # Init current consec request
            last LOITER;                    # Break out LOITER scope
        }

        if (++$CurConsec > $MaxConsec)      # Too many consecutive requests?
        {                                   #   to the same site?
            sleep($PauseTime);              # Yes, take a breather
            $CurConsec = 1;                 # Re-init current consec request
            last LOITER;                    # Break out LOITER scope
        }

        $secs = ($PrevTime + $BetweenTime); # Wake up time in seconds
        if ($secs > time)                   # Not quite time yet to wake up
        {
            sleep($secs - time);            # Sleep the remaining seconds
        }
    }

    $PrevSite = $site;                      # Remember latest site
    $PrevTime = time;                       # Remember current time
}

# ==========================================================================
# Determine from the given $url and response *headers whether it points to a
# "text/html" document. Note that this routine also has the side-effect of
# setting the content-type and content-encoding if they were previously
# undefined.
# Return  1 => text/html content-type
#         0 => not text/html content-type
# ==========================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 339

```
sub is_html
{
    local($url, *headers) = @_;             # URL and http response headers

    if (!defined($headers{'content-type'})) # No content-type specified?
    {
        $_ = $url;                          # Start with URL string
        s#/[^/.]*\.([^/]*)$#/#;             # Grab filename extension off URL
        local($suffix) = ($1 || '');        # Assign suffix string, if any

        # Set content-type and content-encoding headers based on $suffix

        &wwwmime'set_content($suffix, *headers);
    }

    # Match content-type with "text/html" string and return success/failure

    return 1 if ($headers{'content-type'} =~ m#\btext/html\b#io);
    return 0;                               # Not text/html content-type
}

# ============================================================================
# Translate encoded content, of type $encoding and held in $content, into
# its decoded form. The new *content is passed back in place of the old.
# Return  0 => failed to decode content
#         1 => succeeded in decoding content
# ============================================================================
sub decode
{
    local($encoding, *content) = @_;        # Encoding type, encoded content

    return 1 unless ($encoding);            # Return success if not encoded!

    local($com) =  $CEdecoder{$encoding};   # Command for decoding
    local($ext) = ($CEextension{$encoding}  # File name extension
                   || '');                  #   is optional

    return 0 unless ($CEfile && $com);      # Must have file name and command

    if (!open(ENCODE, "> $CEfile$ext"))     # Can't open file for write
    {
        print STDERR "Can't write to $CEfile$ext: $!\n";
        return 0;                           # Failed to decode content
    }
```


IPR2025-00714
DATABRICKS EX1008 Page 340

```
    print ENCODE $content;                  # Write encoded content to file
    close(ENCODE);                          # Close encoded file
    undef $content;                         # Empty all contents

    system("$com $CEfile$ext");             # Execute decoding command
    if (!open(DECODE, $CEfile))             # Can't open file for read
    {
        print STDERR "Can't open decompressed $CEfile: $!\n";
        return 0;                           # Failed to decode content
    }

    local($/);                              # Input record separator
    undef($/);                              # No record separator is matched
    $content = <DECODE>;                    # Read to end of DECODE file
    close(DECODE);                          # Close decoded file
    unlink($CEfile);                        # Remove decoded file

    return 1;                               # Succeeded in decoding content
}

# ==========================================================================
# At the beginning of an infostructure traversal, print out a prologue,
# initialize all counters for statistical summary, and specify values for
# change window and expire window, as appropriate. Also empty all
# information arrays for broken or redirected links, and changed or expired
# nodes. The infostructure is represented in arrays @Task* indexed by $task.
# ==========================================================================
sub begin_summary
{
    local($task) = @_;                      # Current index into @Task*
                                            # Print infostructure prologue
    if ($CGI) { print "<PRE>\n"; }          # HTML formatting as required
    print 'Starting Infostructure [', $TaskName[$task], '] at ',
          &wwwdates'wtime(time, ''), "\n";

    if (defined($TaskChangeWindow[$task]))  # Task change window specified?
    {
        $ChangeDays   = $TaskChangeWindow[$task];
        $ChangeWindow = $TaskChangeWindow[$task] * 86400;
    }
    else { $ChangeDays = $ChangeWindow = 0; }
```


IPR2025-00714
DATABRICKS EX1008 Page 341

```
    if (defined($TaskExpireWindow[$task]))  # Task expire window specified?
    {
        $ExpireDays   = $TaskExpireWindow[$task];
        $ExpireWindow = $TaskExpireWindow[$task] * 86400;
    }
    else { $ExpireDays = $ExpireWindow = 0; }

    &init_summary;                          # Init all counters for summary

    %BrokenNodes   = ();                    # Init broken nodes info array
    %RedirectNodes = ();                    # Init redirected nodes info array
    %ChangedNodes  = ();                    # Init changed nodes info array
    %ExpiredNodes  = ();                    # Init expired nodes info array
}

# ========================================================================
# Initialize all counters for generating statistical table summary results.
# ========================================================================
sub init_summary
{
    $TotalHrefs = 0;                        # Total URL reference count
    $TotalNodes = 0;                        # Total unique node count
    $TotalLocal = 0;                        # Total local node count

    $HrefsTrav  = 0;                        # Traversed URL reference count
    $HrefsTest  = 0;                        # Tested URL reference count
    $HrefsReus  = 0;                        # Reused URL reference count
    $HrefsAvd   = 0;                        # Avoided URL reference count
    $HrefsUnt   = 0;                        # Untestable URL reference count
    $HrefsBroke = 0;                        # Broken URL reference count
    $HrefsRedir = 0;                        # Redirected URL reference count
    $HrefsChg   = 0;                        # Changed URL reference count
    $HrefsExp   = 0;                        # Expired URL reference count
    $HrefsLoc   = 0;                        # Local URL reference count
    $HrefsRmt   = 0;                        # Remote URL reference count

    $NodesTrav  = 0;                        # Unique traversed node count
    $NodesTest  = 0;                        # Unique tested node count
    $NodesReus  = 0;                        # Unique reused node count
    $NodesAvd   = 0;                        # Unique avoided node count
    $NodesUnt   = 0;                        # Unique untestable node count
    $NodesBroke = 0;                        # Unique broken node count
    $NodesRedir = 0;                        # Unique redirected node count
```


IPR2025-00714
DATABRICKS EX1008 Page 342

```
    $NodesChg   = 0;                    # Unique changed node count
    $NodesExp   = 0;                    # Unique expired node count
    $NodesRmt   = 0;                    # Unique remote node count

    $LocalTrav  = 0;                    # Local traversed node count
    $LocalTest  = 0;                    # Local tested node count
    $LocalReus  = 0;                    # Local reused node count
    $LocalAvd   = 0;                    # Local avoided node count
    $LocalUnt   = 0;                    # Local untestable node count
    $LocalBroke = 0;                    # Local broken node count
    $LocalRedir = 0;                    # Local redirected node count
    $LocalChg   = 0;                    # Local changed node count
    $LocalExp   = 0;                    # Local expired node count
}

# ============================================================================
# Determine response message and print on stdout if $node was not $reused.
# Save any info on status of broken or redirected links, and changed nodes,
# as appropriate. Update all direct $Hrefs* counters. The given $node has
# been tested (with 'HEAD' method) as part of the current infostructure.
# ============================================================================
sub tested
{
    local($node, $reused) = @_;         # Tested node, reuse flag
    local(%headers)     = ();           # Headers info to be recalled
    local($url)         = &get_url($node);  # Retrieve URL from history
    local($response)    = &recall($node, *headers);
    local($respmsg);                    # Response message
    local($is_broken)   = 0;            # Is link broken?
    local($is_redirect) = 0;            # Is link redirected?
    local($is_changed)  = 0;            # Is node changed?

    ++$TotalHrefs;                      # Total URL reference count
    if (&is_local($node)) { ++$HrefsLoc; }  # Local URL reference count

    if (&was_avoided($node))            # Node was avoided
    {
        $respmsg = "000 Avoided";       # Replace "000 Unknown Error"
        ++$HrefsAvd;                    # Avoided URL reference count
    }
    elsif (&is_untestable($node))       # Node found to be untestable
    {
        $respmsg = "$RC_not_implemented_client Not Tested";
```


IPR2025-00714
DATABRICKS EX1008 Page 343

```
        ++$HrefsUnt;                        # Untestable URL reference count
    }
    else                                    # All other cases...
    {
        $respmsg = "$response $RespMessage{$response}";
        if (!$reused) { ++$HrefsTest; }     # Tested URL reference count

        local($lmd)  = $headers{'last-modified'};
        local($lmt);                        # Last-modified (secs since Epoch)

        $is_broken   = ($WhatToDo{$response} == $DO_broken);
        $is_redirect = ($WhatToDo{$response} == $DO_redirect);

        $is_changed  = ($ChangeWindow                       &&
                        defined($lmd)                       &&
                        ($lmt = &wwwdates'get_gmtime($lmd)) &&
                        (($lmt + $ChangeWindow) >= time));
    }

    if ($is_broken)                         # Broken link?
    {
        &save_broken($node, $respmsg, $url);
        ++$HrefsBroke;                      # Broken URL reference count
    }
    if ($is_redirect)                       # Redirected link?
    {
        &save_redirect($node, $respmsg, $url);
        ++$HrefsRedir;                      # Redirected URL reference count
    }
    if ($is_changed)                        # Changed node?
    {
        &save_changed($node, $respmsg, $url, $lmd);
        ++$HrefsChg;                        # Changed URL reference count
    }
    if (&is_traversing($node))              # Will or already traversed?
    {
        ++$HrefsTrav;                       # Traversed URL reference count
    }

    print $respmsg if (!$reused);           # Print response message
    print "\n";                             #   if not reused
}
```


IPR2025-00714
DATABRICKS EX1008 Page 344

```
# ============================================================================
# Determine response message and print on stdout. Save any info on status of
# broken or redirected links, changed or expired nodes, as appropriate. The
# given $node has been traversed (with 'GET' method) as part of the current
# infostructure. The node was then (if possible) html-parsed for possible
# child links. The program will next work on testing extracted child links.
# ============================================================================
sub traversed
{
    local($node)        = @_;               # Given node that was traversed
    local(%headers)     = ();               # Headers info to be recalled
    local($url)         = &get_url($node);  # Retrieve URL from history
    local($response)    = &recall($node, *headers);
    local($respmsg);                        # Response message
    local($is_broken)   = 0;                # Is link broken?
    local($is_redirect) = 0;                # Is link redirected?
    local($is_changed)  = 0;                # Is node changed?
    local($is_expired)  = 0;                # Is node expired?

    if (&was_avoided($node))                # Node was avoided
    {
        $respmsg = "000 Avoided";           # Replace "000 Unknown Error"
    }
    elsif (&is_untestable($node))           # Node found to be untestable
    {
        $respmsg = "$RC_not_implemented_client Not Tested";
    }
    else                                    # All other cases...
    {
        $respmsg = "$response $RespMessage{$response}";

        local($lmd)     = $headers{'last-modified'};
        local($expd)    = $headers{'expires'};
        local($lmt);                        # Last-modified (secs since Epoch)
        local($expt);                       # Expiration (secs since Epoch)
        local($current) = time;             # Current time

        $is_broken   = ($WhatToDo{$response} == $DO_broken);
        $is_redirect = ($WhatToDo{$response} == $DO_redirect);

        $is_changed  = ($ChangeWindow                       &&
                        defined($lmd)                       &&
                        ($lmt = &wwwdates'get_gmtime($lmd))  &&
                        (($lmt + $ChangeWindow) >= $current));
```


IPR2025-00714
DATABRICKS EX1008 Page 345

```
        $is_expired  = ($ExpireWindow                          &&
                        defined($expd)                         &&
                        ($expt = &wwwdates'get_gmtime($expd)) &&
                        (($expt - $ExpireWindow) <= $current));
    }

    # Save information arrays for broken or redirected links, as well
    # as for changed or expired nodes, as appropriate

    if ($is_broken)    { &save_broken  ($node, $respmsg, $url);        }
    if ($is_redirect)  { &save_redirect($node, $respmsg, $url);        }
    if ($is_changed)   { &save_changed ($node, $respmsg, $url, $lmd);  }
    if ($is_expired)   { &save_expired ($node, $respmsg, $url, $expd); }

    print $respmsg, "\n";                  # Print response message
}

# ============================================================================
# Save the given $node of broken link along with its response message
# ($respmsg) and $url so that it can later be printed in the list of broken
# links at the end of the infostructure traversal. In case of multiple
# references, prefer the first one saved.
# ============================================================================
sub save_broken
{
    local($node, $respmsg, $url) = @_;

    return if $BrokenNodes{$node};         # Previously saved, just return

    $BrokenNodes{$node} = <<"EOentry";     # Enter in broken nodes array
    $url  ($respmsg)
EOentry

    ++$NodesBroke;                         # Unique broken node count
    ++$LocalBroke if (&is_local($node));   # Local broken node count
}

# ============================================================================
# Save the given $node of redirected link along with its response message
# ($respmsg) and $url so that it can later be printed in the list of
# redirected links at the end of the infostructure traversal. In case of
# multiple references, prefer the first one saved.
# ============================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 346

```
sub save_redirect
{
    local($node, $respmsg, $url) = @_;

    return if $RedirectNodes{$node};        # Previously saved, just return

    $RedirectNodes{$node} = <<"EOentry";    # Enter in redirected nodes array
    $url  ($respmsg)
EOentry

    ++$NodesRedir;                          # Unique redirected node count
    ++$LocalRedir if (&is_local($node));    # Local redirected node count
}

# ============================================================================
# Save the given changed $node along with its response message ($respmsg),
# $url and last-modified date ($lmd) so that it can later be printed in the
# list of changed nodes at the end of the infostructure traversal. In case
# of multiple references, prefer the first one saved.
# ============================================================================
sub save_changed
{
    local($node, $respmsg, $url, $lmd) = @_;

    return if $ChangedNodes{$node};         # Previously saved, just return

    $ChangedNodes{$node} = <<"EOentry";     # Enter in changed nodes array
    $url  ($respmsg)
    Last-modified: $lmd
EOentry

    ++$NodesChg;                            # Unique changed node count
    ++$LocalChg if (&is_local($node));      # Local changed node count.
}

# ============================================================================
# Save the given expired $node along with its response message ($respmsg),
# $url and expiration date ($expd) so that it can later be printed in the
# list of expired nodes at the end of the infostructure traversal. In case
# of multiple references, prefer the first one saved.
# ============================================================================
sub save_expired
{
```


IPR2025-00714
DATABRICKS EX1008 Page 347

```
    local($node, $respmsg, $url, $expd) = @_;

    return if $ExpiredNodes{$node};         # Previously saved, just return

    $ExpiredNodes{$node} = <<"EOentry";     # Enter in expired nodes array
    $url  ($respmsg)
    Expires: $expd
EOentry

    ++$NodesExp;                            # Unique expired node count
    ++$LocalExp if (&is_local($node));      # Local expired node count
}

# ========================================================================
# At the end of traversing an infostructure, print summary of results for
# broken or redirected links, changed or expired nodes, as well as summary
# statistical table, plus an epilogue. The infostructure is represented in
# arrays @Task* indexed by $task.
# ========================================================================
sub end_summary
{
    local($task) = @_;                      # Current index into @TaskName

    if (keys %BrokenNodes)                  # Any broken links?
    {
        print "Broken Links:\n", values(%BrokenNodes);
    }

    if (keys %RedirectNodes)                # Any redirected links?
    {
        print "Redirected Links:\n", values(%RedirectNodes);
    }

    if ((keys %ChangedNodes) &&             # Any changed nodes, and
        ($ChangeWindow > 0))                #   non-zero change window
    {
        print "Changed Links:\n", values(%ChangedNodes);
    }

    if ((keys %ExpiredNodes) &&             # Any expired nodes, and
        ($ExpireWindow > 0))                #   non-zero expiration window?
    {
        print "Expired Documents:\n", values(%ExpiredNodes);
    }
```


IPR2025-00714
DATABRICKS EX1008 Page 348

```
    &update_summary;                        # Update summary table counters
    print "\nSummary of Results:\n", &get_summary;

                                            # Print infostructure epilogue
    print 'Finished Infostructure [', $TaskName[$task], '] at ',
          &wwwdates'wtime(time, ''), "\n\n";
    if ($CGI) { print "</PRE>\n"; }         # HTML formatting as required
}

# ========================================================================
# Update counters used for collecting statistics to generate summary table.
# ========================================================================
sub update_summary
{
    local($node);                           # Index into history arrays

    foreach $node (1 .. $VisNumber)         # For each node visited
    {
        if ($VisInTask[$node])              # URL node is in current task?
        {
            ++$TotalNodes;                  # Increment total node count
            ++$TotalLocal if (&is_local($node));

            if (&was_avoided($node))        # Was URL node avoided?
            {
                ++$NodesAvd;                # Increment avoided node count
                ++$LocalAvd if (&is_local($node));
            }
            elsif (&is_untestable($node))   # Is URL node untestable?
            {
                ++$NodesUnt;                # Increment untestable node count
                ++$LocalUnt if (&is_local($node));
            }
            elsif (&was_tested($node))      # Was URL node tested ('HEAD')?
            {
                ++$NodesTest;               # Increment tested node count
                ++$LocalTest if (&is_local($node));

                if (&was_traversed($node))  # Was URL node traversed ('GET')
                {
                    ++$NodesTrav;           # Increment traversed node count
```


IPR2025-00714
DATABRICKS EX1008 Page 349

```
                ++$LocalTrav if (&is_local($node));
            }
        }
    }
}

    # Total count consists of reused, avoided, untestable and tested counts.
    # Same for URL references, unique URL nodes and local URL nodes.

    $HrefsReus = $TotalHrefs - ($HrefsAvd + $HrefsUnt + $HrefsTest);
    $NodesReus = $TotalNodes - ($NodesAvd + $NodesUnt + $NodesTest);
    $LocalReus = $TotalLocal - ($LocalAvd + $LocalUnt + $LocalTest);

    $HrefsRmt  = $TotalHrefs - $HrefsLoc;   # Total remote URL reference count
    $NodesRmt  = $TotalNodes - $TotalLocal; # Total remote node count
}

# ============================================================================
# Generate statistical summary of results. Return the table-formatted
# statistical summary as an ASCII string ready to be printed out.
# ============================================================================
sub get_summary
{
    local($th) = $TotalHrefs;              # Total URL reference count
    local($tn) = $TotalNodes;              # Total unique URL node count
    local($tl) = $TotalLocal;              # Total local URL node count

    if  (!$th) { $th = 1; }                # Avoid divide by zero errors
    if  (!$tn) { $tn = 1; }
    if  (!$tl) { $tl = 1; }

    return sprintf($SummaryFormat,         # Table-formatted summary
                $HrefsTrav,  (100*$HrefsTrav/$th),
                $NodesTrav,  (100*$NodesTrav/$tn),
                $LocalTrav,  (100*$LocalTrav/$tl),

                $HrefsTest,  (100*$HrefsTest/$th),
                $NodesTest,  (100*$NodesTest/$tn),
                $LocalTest,  (100*$LocalTest/$tl),

                $HrefsReus,  (100*$HrefsReus/$th),
                $NodesReus,  (100*$NodesReus/$tn),
                $LocalReus,  (100*$LocalReus/$tl),
```


IPR2025-00714
DATABRICKS EX1008 Page 350

```
                $HrefsAvd,   (100*$HrefsAvd/$th),
                $NodesAvd,   (100*$NodesAvd/$tn),
                $LocalAvd,   (100*$LocalAvd/$tl),

                $HrefsUnt,   (100*$HrefsUnt/$th),
                $NodesUnt,   (100*$NodesUnt/$tn),
                $LocalUnt,   (100*$LocalUnt/$tl),

                $HrefsBroke, (100*$HrefsBroke/$th),
                $NodesBroke, (100*$NodesBroke/$tn),
                $LocalBroke, (100*$LocalBroke/$tl),

                $HrefsRedir, (100*$HrefsRedir/$th),
                $NodesRedir, (100*$NodesRedir/$tn),
                $LocalRedir, (100*$LocalRedir/$tl),

                $ChangeDays,
                $HrefsChg,   (100*$HrefsChg/$th),
                $NodesChg,   (100*$NodesChg/$tn),
                $LocalChg,   (100*$LocalChg/$tl),

                $ExpireDays,
                $HrefsExp,   (100*$HrefsExp/$th),
                $NodesExp,   (100*$NodesExp/$tn),
                $LocalExp,   (100*$LocalExp/$tl),

                $HrefsLoc,   (100*$HrefsLoc/$th),
                $TotalLocal, (100*$TotalLocal/$tn),
                $TotalLocal,  100.0,

                $HrefsRmt,   (100*$HrefsRmt/$th),
                $NodesRmt,   (100*$NodesRmt/$tn),
                0,              0.0,

                $TotalHrefs,  100.0,
                $TotalNodes, (100*$TotalNodes/$th),
                $TotalLocal, (100*$TotalLocal/$th),
               );
}

# ============================================================================
#                            THE  END                                        #
# ============================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 351

appendix C

WebShopper 1.00 Program Listing

WebShopper is a generic Web agent that can be configured for use as specialized shopping agents, such as BookFinder or CDFinder, that deal only in specific merchandise categories, in this case, books or CDs. WebShopper is designed to assist Web users in searching for and locating comparative items of interest across multiple online shops and to facilitate the painstaking process of comparison shopping based not just on prices alone but also on other terms of sale.

IPR2025-00714
DATABRICKS EX1008 Page 353

The following is a 300-line Perl code listing of WebShopper 1.00, which is also available on the Web at this address:

```
http://deluge.stanford.edu:8000/book/webshop.#!/usr/local/bin/perl

# ============================================================================
# Copyright (c) 1995 Fah-Chun Cheong. All rights reserved.
# ----------------------------------------------------------------------------
# WebShopper 1.00 — A World Wide Web agent for searching multiple online
#                   databases on the Web by author/artist and/or title.
# ============================================================================
#
# WebShopper performs a case insensitive substring search on author/artist
# and title fields. If both the Author/Artist and Title field are filled in,
# the system will display only the items that match on both fields.
#
# ============================================================================

$WebShopper = "BookFinder";                # WebShopper as BookFinder
$webshop    = "bookfind";

# $WebShopper = "CDFinder";                # WebShopper as CDFinder
# $webshop    = "cdfind";

$LibWWW     = "/usr/local/lib/libwww-perl-0.40";
$WWWlib     = ($ENV{'LIBWWW_PERL'} || $LibWWW || '.');
unshift(@INC, $WWWlib);

$Version    = "$WebShopper/1.00";          # Agent identification
$Browser    = "Mozilla/1.1N";              # Favorite browser
$CGIdir     = "/usr/local/www/cgi-bin";

require "getopts.pl";                      # Perl library package
require "www.pl";                          # libwww-perl library package
require "wwwdates.pl";                     # ditto

# ============================================================================
# Set default parameter values.
# ============================================================================

$TaskFile = "$CGIdir/.$webshop";           # default location of task file
$Timeout  = 30;                            # max secs to wait for response
```


IPR2025-00714
DATABRICKS EX1008 Page 354

```
# ============================================================================
# Get the command-line options.
# ============================================================================

if (!(&Getopts('hi:a:t:')) ¦¦ $opt_h) { &usage; }

if ($opt_i) { $TaskFile = $opt_i; }          # Overrides default task file
if ($opt_a) { $Author   = $opt_a; }          # Specify author/artist
if ($opt_t) { $Title    = $opt_t; }          # Specify title

select((select(STDERR), $¦ = 1)[0]);         # Make STDERR unbuffered
$¦ = 1;                                      # Make STDOUT unbuffered

# ============================================================================
# Main processing loop cycling over multiple online databases on the Web.
# ============================================================================

&read_query;                                 # Read query string
&read_tasks;                                 # Read tasks

print $Version, ' starting at ', &wwwdates'wtime(time, ''), "\n";
&www'set_def_header('http', 'User-Agent', $Version);
&www'set_def_header('http', 'User-Agent', $Browser);

&prologue;                                   # Table of contents

foreach $task (1 .. $#TaskName)              # For each specified task
{
    local($content) = '';                    # To hold response content
    &find_item($task, *content);             # Search online database
    &print_result($task, *content);          # Print result of search
}

print $Version, ' finished at ', &wwwdates'wtime(time, ''), "\n";

# ============================================================================
# Print usage information if help requested (-h) or an invalid option given.
# ============================================================================
sub usage
{
    die <<"EndUsage";
usage: $webshop [-h] [-i taskfile] [-a author/artist] [-t title]
$Version
```


IPR2025-00714
DATABRICKS EX1008 Page 355

```
WWW Agent for searching multiple online databases on the Web.
Options:                                                    [DEFAULT]
     -h  Help -- just display this message and quit.
     -i  Get your task instructions from the following file.  [$TaskFile]
     -a  Author/Artist.
     -t  Title.
EndUsage
}

# ============================================================================
# Handle GET and POST method parameters if used as CGI script.
# ============================================================================
sub read_query
{
    local($qstr);                          # query string

    if ($ENV{'REQUEST_METHOD'} eq "GET")    # GET method?
    {
        $qstr = $ENV{'QUERY_STRING'};       # extract query string
    }
    elsif ($ENV{'REQUEST_METHOD'} eq "POST")# POST method?
    {
        read(STDIN, $qstr, $ENV{'CONTENT_LENGTH'});
    }
    else { return $CGI = 0; }               # script not used in CGI context

    $qstr =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack("C",hex($1))/ge;
    $qstr =~ s/\+/ /g;                      # restore blanks in argument values
    local(%env) = split (/[&=]/, $qstr);    # parse into associative array
    local(@env) = split (/&/, $qstr);       # parse into list

    $Author = $env{'Author'};               # extract author/artist
    $Title  = $env{'Title'};                # extract title

    print "Content-type: text/html\n\n";

    return $CGI = 1;                        # script is used in CGI context
}

# ============================================================================
# Read task instructions from the $TaskFile and fill in the @Task* tables.
# ============================================================================
sub read_tasks
{
```


IPR2025-00714
DATABRICKS EX1008 Page 356

```
local($task)   = 0;                    # Task index into @Task* tables
local($intask) = 0;                    # Inside of task description?
local($taskno) = 0;                    # No. of task instructions seen
local($reason) = '';                   # Why task description is bad

if (!$CGI) { print "Reading task specifications from $TaskFile\n\n"; }

if (!open(TASK, $TaskFile))            # Open task file for read
{
    print STDERR "Cannot open task file: $!\n";
    &usage;                            # Show proper usage
}

while (<TASK>)                         # For each line of task file
{
    next if (/^$/ ¦¦ /^\#/);           # Ignore blank and comment lines

    if (!$intask)                      # If not within a task?
    {
        $taskno++;                     # Increment task instruction no.
        if (/^</)                      # Start of next task?
        {
            $task++;                   # New task index
            $intask = 1;               # Now inside a task
        }
        next;                          # Next instructions
    }
                                       # Currently within a task
    if (/^</)                          # Indicates beginning of task?
    {
        die "Task $taskno is not properly terminated, stopped";
    }
    elsif (/^\s*Name\s+((\S+\s*)+)\n$/) { $TaskName  [$task] = $1; }
    elsif (/^\s*Home\s+(\S+)/)          { $TaskHome  [$task] = $1; }
    elsif (/^\s*Form\s+(\S+)/)          { $TaskForm  [$task] = $1; }
    elsif (/^\s*(GET¦POST)\s+(\S+)/)    { $TaskMethod[$task] = $1;
                                          $TaskUrl   [$task] = $2; }
    elsif (/^\s*QUERY\s+(\S+)/)         { $TaskQuery [$task] = $1; }
    elsif (/^\s*AUTHOR\s+(\S+)/)        { $TaskAuthor[$task] = $1; }
    elsif (/^\s*TITLE\s+(\S+)/)         { $TaskTitle [$task] = $1; }
    elsif (/^\s*KEY\s+(\S+)/)           { $TaskKey   [$task] = $1; }
    elsif (/^>\s*$/)                    # Line indicates End of Task?
    {
        $intask = 0;                   # No longer inside task
```


IPR2025-00714
DATABRICKS EX1008 Page 357

```
            $reason = '';                   # No bad task reason until told

            if    (!$TaskName  [$task]) { $reason = "has no Name"; }
            elsif (!$TaskHome  [$task]) { $reason = "has no Home"; }
            elsif (!$TaskForm  [$task]) { $reason = "has no Form"; }
            elsif (!$TaskMethod[$task]) { $reason = "has no Method"; }
            elsif (!$TaskUrl   [$task]) { $reason = "has no Url"; }
            elsif (!($TaskTitle[$task] && $TaskAuthor[$task]) &&
                   !$TaskQuery [$task] &&
                   !$TaskKey   [$task]) { $reason = "has no search string"; }

            if ($reason)                    # If task requirement was not met
            {                               #   then undo its task options
                print(STDERR "Task $taskno ", $reason, ", skipping it.\n");
                undef $TaskName  [$task];
                undef $TaskHome  [$task];
                undef $TaskForm  [$task];
                undef $TaskMethod[$task];
                undef $TaskUrl   [$task];
                undef $TaskQuery [$task];
                undef $TaskAuthor[$task];
                undef $TaskTitle [$task];
                undef $TaskKey   [$task];
                $task--;                    # Bad task don't count
            }
        }
        else
        {
            print STDERR "Unrecognized option in task $taskno\n";
            print STDERR "  at line $. of $TaskFile\n";
        }
    }

    if ($intask)                            # Inside task but end of file?
    {
        die "Last instruction is not properly terminated, stopped";
    }
    close TASK;                             # Close task file
}

# ============================================================================
# Print summarizing table of contents in a prologue.
# ============================================================================
sub prologue
```


IPR2025-00714
DATABRICKS EX1008 Page 358

```
{
    print "<hr><H1>Results from $WebShopper</H1>\n";
    print "<H2>The following online databases was searched:</H2>\n";
    print "<ul>\n";

    foreach $task (1 .. $#TaskName)         # For each specified task
    {
        print "<li><a href=\"$TaskHome[$task]\">$TaskName[$task]</a>\n";
    }
    print "</ul>\n";
    print "<hr size=8>\n";
}

# ========================================================================
# Make all URL references in $content absolute (and not relative).
# ========================================================================
sub globalize
{
    local(*content, $url) = @_;

    $content =~ s/\<A(.*)\s+HREF\s*=\s*([^\"])([^\s\>]*)/\<A$1
    HREF=\"$url$2$3\"/
gi;
    $content =~ s/\<IMG(.*)\s+SRC\s*=\s*([^\"])([^\s\>]*)/\<IMG$1
    SRC=\"$url$2$3\
"/gi;
    $content =~ s/\<FORM(.*)\s+ACTION\s*=\s*([^\"])([^\s\>]*)/\<FORM$1
    ACTION=\"$
url$2$3\"/gi;

    $content =~
    s/\<A(.*)\s+HREF\s*=\s*(\"?)(\/[^\/]\S*¦\/¦\.\S*)(\"?[^\>]*)\>/\<
A$1 HREF=$2$url$3$4\>/gi;
    $content =~
    s/\<IMG(.*)\s+SRC\s*=\s*(\"?)(\/[^\/]\S*¦\/¦\.\S*)(\"?[^\>]*)\>/\
<IMG$1 SRC=$2$url$3$4\>/gi;
    $content =~
    s/\<FORM(.*)\s+ACTION\s*=\s*(\"?)(\/[^\/]\S*¦\/¦\.\S*)(\"?[^\>]*)
\>/\<FORM$1 ACTION=$2$url$3$4\>/gi;
}

# ========================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 359

```
# Search the given database indicated by $task, return result in *content.
# ============================================================================
sub find_item
{
    local($task, *content) = @_;            # Result of $task in *content
    local($key)     = $Title ¦¦ $Author;    # Build keyword
    local(%headers) = ();                   # To hold parsed response headers
    local($headers) = '';                   # To hold response headers
    local($url);                            # URL to GET or POST
    local($response);                       # Response code returned

    $TaskAuthor[$task] =~ s/\$Author/$Author/;
    $TaskTitle [$task] =~ s/\$Title/$Title/;
    $TaskQuery [$task] =~ s/\$Author/$Author/;
    $TaskQuery [$task] =~ s/\$Title/$Title/;
    $TaskKey   [$task] =~ s/\$Key/$key/;

    if ($Author && $Title)                  # Search by both author and title
    {
        $content = $TaskQuery [$task] ¦¦
                   $TaskKey   [$task] ¦¦
                      $TaskTitle [$task] ¦¦
                   $TaskAuthor[$task];
    }
    elsif ($Author)                         # Search by author
    {
        $content = $TaskQuery [$task] ¦¦
                   $TaskAuthor[$task] ¦¦
                    $TaskKey   [$task];
    }
    elsif ($Title)                          # Search by title
    {
        $content = $TaskQuery [$task] ¦¦
                    $TaskTitle [$task] ¦¦
                   $TaskKey   [$task];
    }
    else { return; }                        # Skip

    if ($CGI)
    {
        $headers{'Accept'} = '*/*';
        $headers{'Accept'} = 'image/x-xbitmap';
        $headers{'Accept'} = 'image/gif';
```


IPR2025-00714
DATABRICKS EX1008 Page 360

```
        $headers{'Accept'} = 'image/jpeg';
    }
    else
    {
        $headers{'Accept'} = '*/*';
    }
    $headers{'Content-type'}   = 'application/x-www-form-urlencoded';
    $headers{'Content-length'} = length($content);

    $url = $TaskMethod[$task] eq "GET"      # GET method?
         ? "$TaskUrl[$task]?$content"       # Concat search string
         :  $TaskUrl[$task];                # Leave intact

    $response = &www'request($TaskMethod[$task], $url,
                             *headers, *content, $Timeout);

                                            # Make partial URL global

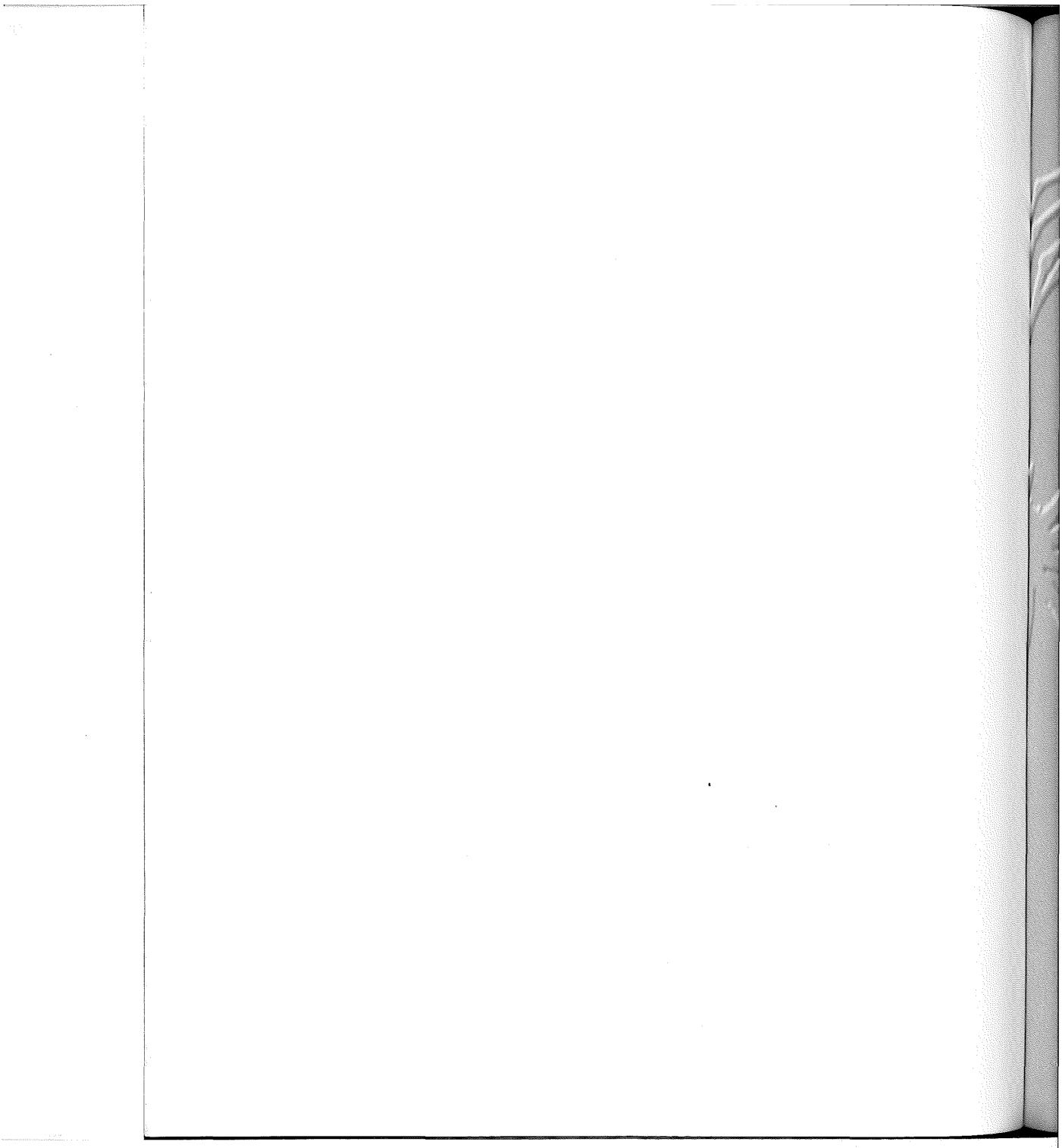
    $url =~ s/http:\/\/([^\/]+)\/.*$/http:\/\/$1/gi;
    &globalize(*content, $url);
}

# ==========================================================================
# Print result of searching the online database identified by $task.
# ==========================================================================
sub print_result
{
    local($task, *content) = @_;            # Result of $task in *content

    print "<H1><I><a name=#tag$task>";      # HTML text fragment
    print "$TaskName[$task]</a></I></H1><p>\n";
    print $content;                         # Result of searching database
    print "<hr size=8>\n";                  # Result delimiter
}

# ==========================================================================
1; #                              THE  END                                  #
# ==========================================================================
```


IPR2025-00714
DATABRICKS EX1008 Page 361



appendix D

List of Online Bookstores Visited by BookFinder

The BookFinder agent is a specialized instance of WebShopper that helps users locate and buy books from bookstores on the Web. For testing purposes, a sample BookFinder agent has been set up to run from `http://deluge.stanford.edu:8000/book/bookfind.html` (see fig. D.1).

IPR2025-00714
DATABRICKS EX1008 Page 363

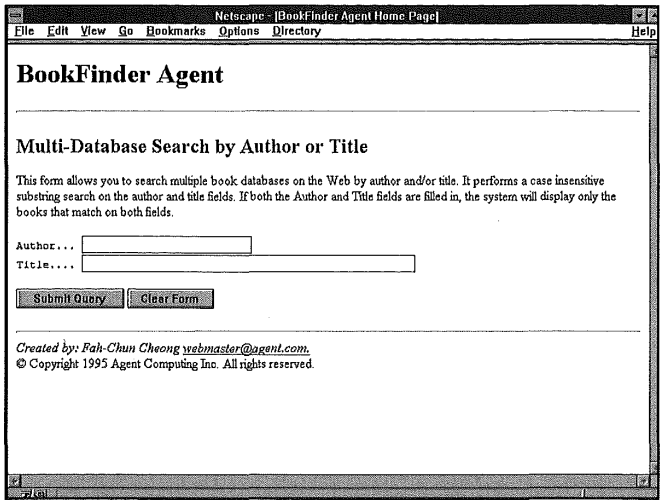


Figure D.1

The sample BookFinder agent home page.

Users are encouraged to download the WebShopper code from `http://deluge.stanford.edu:8000/book/webshop` and run it directly on their own machines after configuring the WebShopper to behave as BookFinder according to instructions contained therein. The following is a list of online bookstores on the Web that are visited by BookFinder agents (and should appear in a default task file named ".bookfind").

Name	Book Stacks Unlimited
Home	`http://www.books.com/`
Form	`http://melville.books.com/scripts/search1.exe?sid=gXnbjdMaDbGxYye`
POST	`http://shaker.books.com/scripts/search.exe?sid=gXnbjdMaDbGxYye`
TITLE	`type=title&search=$Title`
AUTHOR	`type=author&search=$Author`

Name	CompuBooks
Home	`http://www.compubooks.com/books.html`
Form	`http://www.compubooks.com/order/books/search`
POST	`http://www.compubooks.com/order/books/findit/2454`
QUERY	`author=$Author&title=$Title&file=1.+All+Types`

Name	Computer Literacy Bookshops
Home	`http://www.clbooks.com/`
Form	`http://www.clbooks.com/cgi-bin/browsedb`
POST	`http://www.clbooks.com/cgi-bin/browsedb?20016`

IPR2025-00714
DATABRICKS EX1008 Page 364

```
TITLE     SEARCH.STRING=$Title&SEARCH.TYPE=TITLE&SEARCH.ALG=Basic+Search
AUTHOR    SEARCH.STRING=$Author&SEARCH.TYPE=AUTHOR&SEARCH.ALG=Basic+Search

Name      Future Fantasy Bookstore
Home      http://futfan.com/home.html
Form      http://futfan.com/search.html
POST      http://futfan.com/cgi-bin/query
QUERY     author=$Author&title=$Title

Name      Intertain.com
Home      http://intertain.com/store/welcome.html
Form      http://intertain.com/store/search.html
POST      http://intertain.com/cgi-bin/loc_query
QUERY     catalog=non-fiction&type=INDEX&author=$Author&title=$Title

Name      Macmillan Bookstore
Home      http://www.mcp.com/cgi-bin/do-bookstore.cgi
Form      http://www.mcp.com/cgi-bin/do-searches.cgi?userID=46345811
POST      http://www.mcp.com/cgi-bin/find-books.cgi
TITLE     search_term=$Title&by_some=by_title&and_or=
          as_and&userID=46345811
AUTHOR    search_term=$Author&by_some=by_author&and_or=
          as_and&userID=46345811

Name      Powell's Technical Books
Home      http://www.technical.powells.portland.or.us/
Form      http://www.powells.portland.or.us/search.technical.html
POST      http://www.powells.portland.or.us/cgi-bin/search.pl
QUERY     title-strings=$Title&author-strings=$Author

Name      Reiters Books
Home      http://www.awa.com/reiters/index.html
Form      http://www.awa.com/reiters/haveyou.reiters.html
POST      http://www.awa.com/cgi-bin/grepper.reiters.pl
KEY       searchword=$Key

Name      Roswell Internet Computer Bookstore
Home      http://www.nstn.ns.ca/cybermall/roswell/roswell.html
Form      http://www.nstn.ca/cgi-bin/roswell-engine?search
POST      http://www.nstn.ca/cgi-bin/roswell-engine?search
KEY       searchtext=$Key&MFT=search
```


IPR2025-00714
DATABRICKS EX1008 Page 365

```
Name    Softpro Books
Home    http://internet-plaza.net/softpro/
Form    http://internet-plaza.net/cgi-bin/softpro-booklist-agrep.pl
GET     http://internet-plaza.net/cgi-bin/softpro-booklist-agrep.pl
KEY     $Key
```


IPR2025-00714
DATABRICKS EX1008 Page 366

appendix E

List of Online Music Stores Visited by CDFinder

The CDFinder agent is a specialized instance of WebShopper that helps users locate and buy CD albums from music stores on the Web. For testing purposes, a sample CDFinder agent has been set up to run from `http://deluge.stanford.edu:8000/book/cdfind.html`.

IPR2025-00714
DATABRICKS EX1008 Page 367

Users are encouraged to download the WebShopper code from `http://deluge.stanford.edu:8000/book/webshop` and run it directly on their own machines after configuring the WebShopper to behave as CDFinder according to instructions contained therein. The following is a list of online music stores on the Web that are visited by CDFinder agents (and should appear in a default task file named ".cdfind").

```
Name    CD Land
Home    http://www.persimmon.com/CDLand/
Form    http://www.persimmon.com/CDLand/cd-first.html
POST    http://www.persimmon.com/cgi-bin/picard
QUERY   repeat=0&format=/usr/local/HTML_docs/CDLand
        /cdland.format&picard-search-t=$Title&picard-search-a=
        $Author&picard-select-media=ALL&match-all=match-all

Name    CDNow
Home    http://cdnow.com/
Form    http://cdnow.com:445/cgi-bin/mserver/SID=890838739/page=popfloor
POST    http://cdnow.com:445/cgi-bin/mserver/SID=890838739/page=popsearch
TITLE   index=t&string=$Title&order=A&quick=Y
AUTHOR  index=a&string=$Author&order=A&quick=Y

Name    CD World
Home    http://cdworld.com/
Form    http://cdworld.com/cgi-bin/nav?cdfind&421023
POST    http://cdworld.com/cgi-bin/cdfind1
QUERY   maxhits=25&title=$Title&artist=$Author

Name    Global Electronic Music Marketplace
Home    http://192.215.9.13/Flirt/GEMM/gemmlean.c.html
Form    http://192.215.9.13/Flirt/GEMM/GEMM_HTTP/search/search1_form.html
POST    http://gemm.com/gemm.acgi
QUERY   S=Submit&Artist=$Author&Title=$Title&Genre=
        %28ANY%29&Media=CD&MaxNRecs=25&priceLow=0&priceHigh=100

Name    i? music/media
Home    http://www.icw.com/cd/imm1.html
Form    http://www.icw.com/cd/datab.html
POST    http://www.icw.com/cgi-bin/cdwais
KEY     name=$Key
```


IPR2025-00714
DATABRICKS EX1008 Page 368

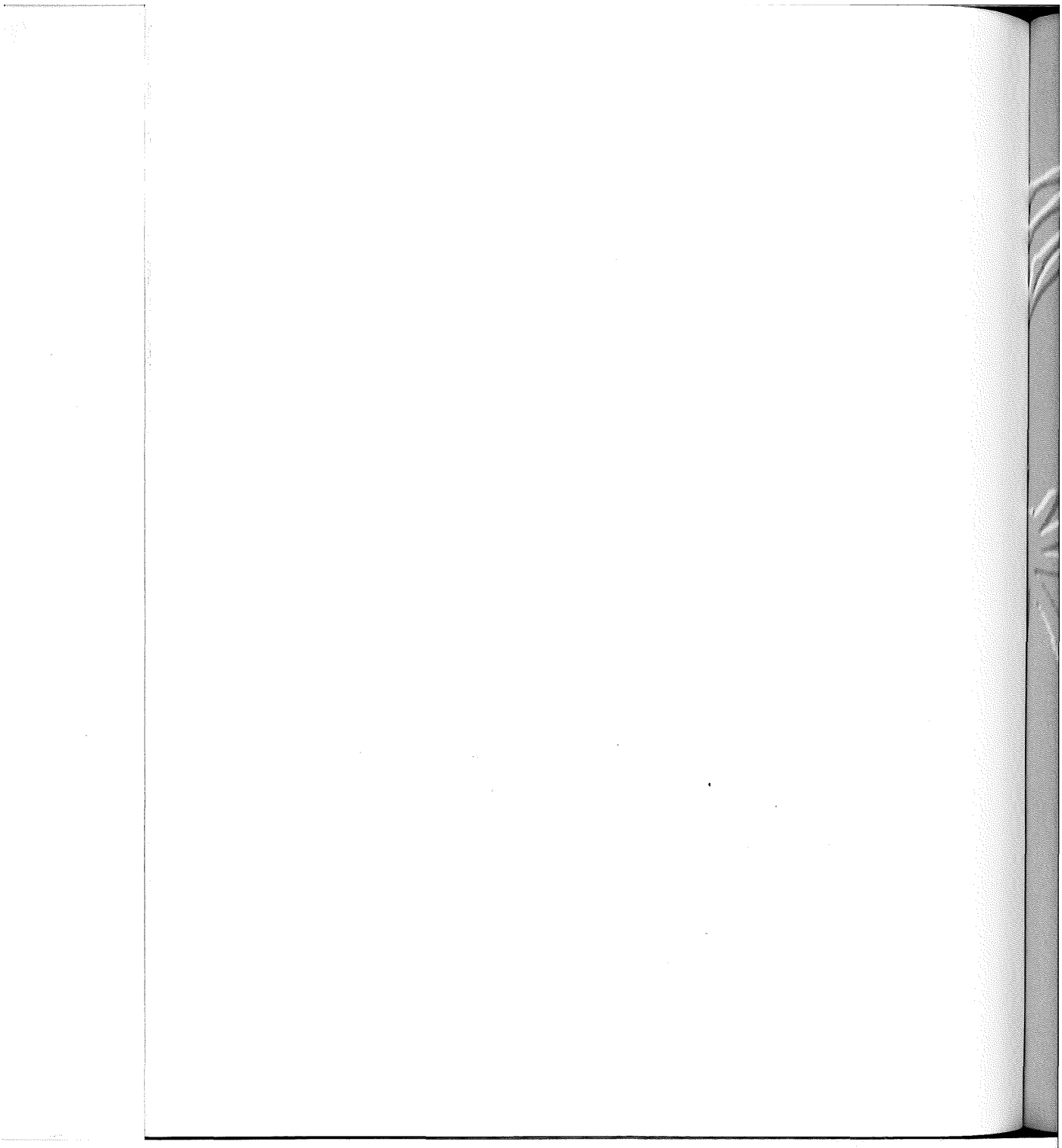
```
Name      Music Connection
Home      http://www.inetbiz.com/music/
Form      http://www.inetbiz.com/cgi-bin action?type=static&page=search.html&state
          =000013c2
GET       http://www.inetbiz.com/cgi-bin/action
QUERY     type=fastscan&state=000013c2&artist=$Author&title=
          $Title&category=Rock%2C+Pop%2C+R+%26+B

Name      Noteworthy Music
Home      http://www.netmarket.com/noteworthy/bin/main/:grm¦:mode=text
Form      http://www.netmarket.com/noteworthy/bin/searchsimple/
          :st=6ou5fui852¦1
POST      http://www.netmarket.com/noteworthy/bin/searchresult/
          :st=6ou5fui852¦2/:t=52474638/:page=1
TITLE     :sm_1=t&:ss_1=$Title
AUTHOR    :sm_1=a&:ss_1=$Author

Name      Tower Records
Home      http://www.shopping2000.com/shopping2000/tower/
Form      http://www.shopping2000.com/shopping2000/tower/
POST      http://www.shopping2000.com/cgi-bin/tower-search.pl
KEY       query=$Key

Name      EMusic
Home      http://emusic.com/
Form      http://emusic.com/browse
POST      http://emusic.com/browse/1272107
TITLE     idx=t&text=$Title
AUTHOR    idx=a&text=$Author
```


IPR2025-00714
DATABRICKS EX1008 Page 369



appendix F

List of Active MUD Sites on the Internet

This is Doran's MUDlist, which contains most of the known sites where MUDs of different types are running. It is maintained by Adam Wozniak at awozniak@geordi.calpoly.edu, also known as Doran.

IPR2025-00714
DATABRICKS EX1008 Page 371

Doran's MUDlist is published every once in a while and posted to the Usenet newsgroup rec.games.mud.announce. It is available on the World Wide Web at `http://www.cm.cf.ac.uk/User/Andrew.Wilson/MUDlist/` or it can be received via e-mail by sending a message to `mudlist@lore.calpoly.edu`.

Doran's MUDlist

TYPE <unknown> (2)

CyberEden	`rivendel.slip.umd.edu`	128.8.11.201	5000
Formosa	`db82.csie.ncu.edu.tw`	140.115.50.82	4000

TYPE aber (19)

An old hack-and-slash style MUD.

BabeMUD	`teaching4.physics.ox.ac.uk`	163.1.245.204	4001
BernieMUD	`fwk098037.res-hall.nwu.edu`	199.74.98.37	6715
Budapest	`prometheus.bsd.uchicago.edu`	128.135.30.5	6789
DragonMUD	`fermina.informatik.rwth-aachen.de`	137.226.224.20	6715
Eclipse MUD	`mud.bsd.uchicago.edu`	128.135.90.79	6715
Infinity	`sirius.nmt.edu`	129.138.4.119	6715
Kender's Kove	`harvey.esu.edu`	192.148.218.99	6715
Mustang	`mustang.us.dell.com`	143.166.224.42	9173
Northern Lights	`aber.ludd.luth.se`	130.240.16.29	6715

IPR2025-00714
DATABRICKS EX1008 Page 372

PuRgAtorY	purgatory.jaxnet.com	199.170.71.2	2112
Rainbow	aber.nhidh.nki.no	128.39.98.162	6715
Silver	dante.exide.com	198.85.1.1	6715
Sleepless Nights	cs3.brookes.ac.uk	161.73.1.2	6789
Terradome	cmssrv-gw.brookes.ac.uk	161.73.101.20	8888
Tyrann II	muselab-gw.runet.edu	137.45.128.10	6715
Tyrann II	muselab.ac.runet.edu	137.45.33.100	6715
Ulrik/Dirt	alkymene.uio.no	129.240.21.60	6715
Utopia	alpha.dsu.edu	138.247.32.12	6789
WhirlWind	bubba.ucc.okstate.edu	139.78.100.14	6715

TYPE almostIRC (1)

No information available.

Olohof Bbs	morra.et.tudelft.nl	130.161.144.100	2993

TYPE bsx (1)

A graphical hack-and-slash MUD. Special client required.

Regenesis	birka.lysator.liu.se	130.236.254.159	7475

TYPE circle (6)

A hack-and-slash style MUD.

AbsolutMUD	b63740.cwru.edu	129.22.248.252	4000
BetaMud	thijm.si.hhs.nl	145.52.81.25	4000
HexOnyx	marble.bu.edu	128.197.10.75	7777
LoveMUD	mud.butterfly.net	199.73.40.4	4000

IPR2025-00714
DATABRICKS EX1008 Page 373

PrimalMud	jeack.apana.org.au	202.12.87.82	4000
Virtual World of Magma	magma.leg.ufrj.br	146.164.53.33	4000

TYPE darkmud (1)

No information available.

Midian][	pandora.mit.csu.edu.au	137.166.16.2	3333

TYPE dgd (6)

A hack-and-slash style MUD very similar to lp.

Albion Mud	veda.is	193.4.230.1	4000
GodsHome	godshome.solace.mh.se	193.10.118.131	3000
IgorMUD	igor.mtek.chalmers.se	129.16.60.113	1701
MirrorMOO	mirror.ccs.neu.edu	129.10.112.76	8889
PaderMUD	mud.uni-paderborn.de	131.234.12.13	3000
The Pattern	epsilon.me.chalmers.se	129.16.50.30	6047

TYPE diku (57)

A hack-and-slash style MUD.

AbsolutMUD	b63740.STUDENT.CWRU.Edu	129.22.248.252	4000
Abyss II	helpmac.its.mcw.edu	141.106.32.113	8888
Albanian	fred.indstate.edu	139.102.12.14	2150
AlexMUD	marcel.stacken.kth.se	130.237.234.17	4000
Apocalypse IV	sapphire.geo.wvu.edu	157.182.168.20	4000
AustinMud	imv.aau.dk	130.225.2.2	4000
Chaos	chaos.bga.com	198.3.118.12	4000

IPR2025-00714
DATABRICKS EX1008 Page 374

Dark Castle Mud	jive.rahul.net	192.160.13.4	6666
DikuMUD	mud.stacken.kth.se	130.237.234.17	4000
DikuMud II	marcel.stacken.kth.se	130.237.234.17	4242
Elite	xbyse.nada.kth.se	130.237.222.237	4000
Empire	einstein.physics.drexel.edu	129.25.1.120	4000
FieryMud	fiery.eushc.org	163.246.5.109	4000
* Forbidden Lands	queen.mcs.drexel.edu	129.25.7.100	4000
FuskerMud	hopi.dtcc.edu	138.123.76.240	4000
Games of Death (G.O.D.) MUD	cyberspace.com	199.2.48.12	4000
GrimneMUD	gytje.pvv.unit.no	129.241.36.226	4000
Hercules MUD	sunshine.eushc.org	163.246.32.110	3000
Imperial	supergirl.cs.hut.fi	130.233.40.52	6969
JeenusTooMUD	heegaard.mth.pdx.edu	131.252.40.91	4000
KAOS HQ	sg25.aud.temple.edu	129.32.66.7	4000
KallistiMud	jadzia.CSOS.ORST.EDU	128.193.40.23	4000
The Land	nora.gih.no	128.39.140.150	4000
Legend of the Winds	ccsun44.csie.nctu.edu.tw	140.113.17.168	4040
MEDIEVIA Cyberspace	intense.netaxs.com	198.69.186.36	4000
MUME	lbdsun4.epfl.ch	128.178.77.5	4242
Mayhem	thrash.isca.uiowa.edu	128.255.200.25	1234

IPR2025-00714
DATABRICKS EX1008 Page 375

Meat MUD	sneezy.cc.utexas.edu	128.83.108.16	2800
Medievia Cyberspace	medievia.netaxs.com	198.69.186.36	4000
MooseHead Mud	mud.eskimo.com	204.122.16.44	4000
Mozart	kitten.mcs.com	192.160.127.90	4500
# Mudde Pathetique	flysex.berkeley.edu	128.32.128.36	2999
NaVie	mud.cs.odu.edu	128.82.6.30	4000
# NexusMUD	didec3.epfl.ch	128.178.164.5	4000
Northern Crossroads	ugsparc13.eecg .toronto.edu	128.100.13.63	9000
OpalMUD	opal.cs.virginia.edu	128.143.60.14	4000
PK MUD	kennedy.ecn.uoknor.edu	129.15.24.14	5000
Perilous Realms	PR.mese.com	155.229.1.4	23
Phantazm	fpa.com	198.242.217.1	4000
Realm of Magic	p107.informatik .uni-bremen.de	134.102.216.8	4000
RockyMud	hermes.dna.mci.com	166.41.48.146	4000
RoninMUD	ronin.bchs.uh.edu	129.7.2.127	5000
Shadowdale	dale.hsc.unt.edu	129.120.104.40	7777
Silicon Realms	sampan.ee.fit.edu	163.118.30.9	4000
SillyMUD	stone.cis.ufl.edu	128.227.100.197	4000
SlothMUD II	ai.cs.ukans.edu	129.237.80.113	6101
Sojourn	sojourn.cem.msu.edu	35.8.25.23	9999
# StrangeMUD	sleepy.cc.utexas.edu	128.83.108.13	9332

IPR2025-00714
DATABRICKS EX1008 Page 376

StrikeNet	falcon.depaul.edu	140.192.1.7	4000
Tazmania	ukko.rowan.edu	150.250.10.216	5000
TemporaryMUD	cc.joensuu.fi	128.214.14.12	4000
The Final Level	huey.ee.cua.edu	136.242.140.31	7777
The Glass Dragon	surf.tstc.edu	161.109.32.2	4000
ThunderDome	tdome.montana.com	199.2.139.3	5555
Wayne's World	drake.eushc.org	163.246.32.100	9000
WildSide	levant.cs.ohiou.edu	132.235.1.100	1234
Worlds of Carnage	orcrist.digital.ufl.edu	128.227.133.214	4000

TYPE dum (3)

No information available.

CanDUM II	itrchq.itrc.on.ca	128.100.3.100	2001
DUM II	dum.ludd.luth.se	130.240.16.36	2001
FranDUM II	mousson.enst.fr	137.194.160.48	2001

TYPE lp (111)

A hack-and-slash style MUD.

3-Kingdoms	marble.bu.edu	128.197.10.75	5000
5th Dimension	gauss.ifm.liu.se	130.236.50.9	3000
Alatia	neumark.physik.tu-magdeburg.de	141.44.40.24	3000
AlexMUD	mud.stacken.kth.se	130.237.234.17	4000
Ancient Anguish	dancer.ethz.ch	129.132.57.66	2222
Ancient Anguish	end2.bedrock.com	131.158.153.39	2222

IPR2025-00714
DATABRICKS EX1008 Page 377

Angalon	neuromancer.tamu.edu	128.194.47.9	3011
Asteroth (LPSwat)	aviator.cc.iastate.edu	129.186.140.6	2020
Aurora	aurora.etsiig.uniovi.es	156.35.41.20	3000
BatMUD	bat.cs.hut.fi	130.233.40.180	23
BatMUD	palikka.jyu.fi	130.234.40.3	2001
Callandor	warns.et.tudelft.nl	130.161.147.41	5317
Castalia	miucs1.miu.edu	192.103.45.2	4444
Chatter	hawking.u .washington.edu	140.142.58.99	6000
Conservatorium	crs.cl.msu.edu	35.8.1.10	6000
Coupworld	honey.st-and.ac.uk	138.251.33.32	7425
Crossed Swords	shsibm.shh.fi	128.214.44.251	3000
Dark Saga	cobber.cord.edu	138.129.1.32	5555
DarkWind	darkwind.i-link.com	198.67.37.33	3000
Darker Realms	worf.tamu.edu	128.194.116.25	2000
Dartmud	fermat.dartmouth.edu	129.170.28.31	2525
Deeper Trouble	alk.iesd.auc.dk	130.225.48.46	4242
Discworld	cix.compulink.co.uk	192.188.69.2	4242
Dragon's Den	hellfire.dusers .drexel.edu	129.25.56.246	2222
Dragon's Den	oyster01.mcs.drexel.edu	129.25.7.111	2222
DragonFire	typo.umsl.edu	134.124.42.197	3000
Dshores	kcbbs.gen.nz	202.14.102.1	6000
Eastern Stories	cisppc2.cis.nctu.edu.tw	140.113.204.42	8000

IPR2025-00714
DATABRICKS EX1008 Page 378

Elements of Paradox	elof.acc.iit.edu	192.41.245.90	6996
Enigma	kcbbs.gen.nz	202.14.102.1	6000
Etheria	csgi60.leeds.ac.uk	129.11.144.190	7777
EverDark	atomic.com	198.64.6.24	3000
Final Realms	kark.hiof.no	158.36.33.5	2001
First Light	gold.t-informatik .ba-stuttgart.de	141.31.1.16	3000
GateWay	idiot.alfred.edu	149.84.4.1	6969
Genesis	hamal2.cs.chalmers.se	129.16.226.142	3011
Genocide	pip.shsu.edu	192.92.115.10	2222
Gilgamesh	shiner.st.usm.edu	131.95.115.131	3742
# Hall of Fame	marvin.df.lth.se	194.47.62.1	2000
HariMUD	tc0.chem.tue.nl	131.155.94.3	6997
Haven	idrz07.ethz.ch	129.132.76.8	1999
* Holy Mission	alijku05.edvz .uni-linz.ac.at	140.78.3.1	2001
Igor	ny.mtek.chalmers.se	129.16.60.9	1701
Imperial2	hp.ecs.rpi.edu	128.113.5.43	3141
Infinity	olympus.ccs.neu.edu	129.10.111.80	3000
Ivory Tower	marvin.macc.wisc.edu	144.92.30.207	2000
Kerovnia	atlantis.edu	192.67.238.48	1984
Kingdoms	gwaihir.dd.chalmers.se	129.16.117.21	1812
* Kingdoms	kili.dd.chalmers.se	129.16.117.17	1812
KoBra	kobra.et.tudelft.nl	130.161.144.236	23

IPR2025-00714
DATABRICKS EX1008 Page 379

Kobra	duteca4.et.tudelft.nl	130.161.144.22	8888
LOST Mud	louie.cc.utexas.edu	128.83.108.12	6666
Last Outpost	lo.millcomm.com	199.170.133.6	4000
Loch Ness	indigo.imp.ch	157.161.1.12	2222
LooneyMud	looney.cp.tn.tudelft.nl	192.31.126.102	8888
Lost Souls	ronin.bchs.uh.edu	129.7.2.127	3000
# LustyMud	lusty.tamu.edu	128.194.9.199	2000
Marches of Antan	checfs2.ucsd.edu	132.239.68.9	3000
Mathmud	medusa.math .fu-berlin.de	130.133.4.4	3000
Midnight Sun	holly.ludd.luth.se	130.240.16.23	3000
Moonstar	pulsar.hsc.edu	192.135.84.5	4321
Muddog	catalyst.math.ufl.edu	128.227.168.38	2000
Muddy Waters	hot.caltech.edu	131.215.9.49	3000
Mythos	mann.uni-koblenz.de	141.26.4.41	3000
NANVAENT 3	corrour.cc.strath.ac.uk	130.159.220.8	3000
NannyMUD	birka.lysator.liu.se	130.236.254.159	2000
Newmoon	jove.cs.pdx.edu	131.252.21.12	7680
* NightFall	nova.tat.physik .uni-tuebingen.de	134.2.62.161	4242
Nirvana IV	elof.acc.iit.edu	192.41.245.90	3500
Nuclearwar	Melba.Astrakan.HGS.SE	130.238.206.12	23
Overdrive	castor.acs.oakland.edu	141.210.10.109	5195
OzRogue	mud.geko.com.au	203.2.239.3	5001

IPR2025-00714
DATABRICKS EX1008 Page 380

OzRogue	mud.geko.com.au	203.2.239.3	5002
Paradox	adl.uncc.edu	152.15.15.18	10478
Phoenix	albert.bu.edu	128.197.74.10	3500
PixieMud	elof.acc.iit.edu	192.41.245.90	6969
Pixilated	elof.acc.iit.edu	192.41.245.90	6996
Prime Time	prime.mdata.fi	192.98.43.2	3000
Prydein	spock.austin.apple.com	204.96.16.2	4567
Psycho-thriller	atlantis.edu	192.67.238.48	3000
Pyromud	elektra.cc.edu	140.104.1.100	2222
Ragnarok	ragnarok.teleport.com	192.108.254.22	2222
Realms of the Dragon	cw-u04.umd.umich.edu	141.215.69.7	3000
Realmsmud	donal.dorsai.org	198.3.127.6	1501
Regenesis	regenesis.lysator .liu.se	130.236.254.159	7475
# Regula	rrzc8.rz .uni-regensburg.de	132.199.30.8	3000
Renegade Outpost	outpost.hsc.unt.edu	129.120.104.19	9999
Revenge of the End of the Line	aus.Stanford.edu	36.21.0.99	2010
STYX	dreamtime.nmsu.edu	128.123.8.116	3000
SWmud	kitten.mcs.net	192.160.127.90	6666
SplitSecond	lestat.shv.hb.se	130.241.246.10	3000
Starmud	krynn.solace.mh.se	193.10.118.130	7373
Starmud	starmud.solace.mh.se	193.10.118.130	4000

IPR2025-00714
DATABRICKS EX1008 Page 381

StickMUD	kalikka.jyu.fi	130.234.40.2	7680
StickMUD	lancelot.cc.jyu.fi	130.234.40.4	7680
T'Mud	wave.st.usm.edu	131.95.119.2	2222
TAPPMud	surprise.pro.ufz.de	141.65.31.2	6510
TMI-2	kendall.ccs.neu.edu	129.10.114.86	5555
# The Holy Mission	alijku03.edvz .uni-linz.ac.at	140.78.3.30	4242
Timewarp	quark.gmi.edu	192.138.137.39	5150
Tron	polaris.king.ac.uk	141.241.84.65	3000
Tron	polaris.king.ac.uk	141.241.84.65	4000
TubMUD	morgen.cs.tu-berlin.de	130.149.19.20	7680
Ultimate	rkw-risc.cs.up.ac.za	137.215.18.10	1984
Valhalla	valhalla.com	192.187.153.1	2444
VikingMUD	sid.dsl.unit.no	129.241.36.75	2001
Windy	bitsy.apana.org.au	203.2.134.3	2000
Wonderland	gorina3.hsr.no	152.94.1.43	3287
Zebedee	rszircon.swan.ac.uk	137.44.102.2	7000
ZombieMUD	linux1.sjoki.uta.fi	192.98.84.5	3333
# flame	marlin.ucc.gu .uwa.edu.au	130.95.100.4	4242

TYPE lp-german (3)

A hack-and-slash style lp MUD, but in German!

MorgenGrauen	PASCAL.UNI-MUENSTER.DE	128.176.121.56	4711
SvenskMUD	bodil.lysator.liu.se	130.236.254.152	2043

IPR2025-00714
DATABRICKS EX1008 Page 382

UNITOPIA	helpdesk.rus .uni-stuttgart.de	129.69.221.120	3333

TYPE mare (1)

An adventure role-playing MUD.

WindsMare	cyberion.bbn.com	128.89.2.139	7348

TYPE merc (13)

A diku spinoff hack-and-slash MUD.

Barren Realms	liii.com	198.207.193.1	8000
Black Knights Legacy	peseta.unice.fr	193.48.228.102	6000
Dragons, Legends, and Lore	trident.ee.fit.edu	163.118.30.14	9000
Farside	farside.atinc.com	198.138.35.199	3000
Hidden Worlds	cns.cscns.com	192.156.196.1	4000
Highlands	jedi.cis.temple.edu	129.32.32.70	9001
Legends of Winds	ccsun44.csie .nctu.edu.tw	140.113.17.168	4040
Lost Realms	viper.ucs.uoknor.edu	129.15.10.23	4000
MadROM	dogbert.ugcs .caltech.edu	131.215.133.151	1536
Mortal Realms	hydrogen.ee.utulsa.edu	129.244.42.48	4321
StackMUD	marcel.stacken.kth.se	130.237.234.17	8000
Stick in the MUD	ugsparc11.eecg .utoronto.ca	128.100.13.61	9000
TIAC Mud	toybox.tiac.net	199.0.65.25	9999

IPR2025-00714
DATABRICKS EX1008 Page 383

TYPE moo (20)

A tiny spinoff chat MUD with an object-oriented internal programming language.

BayMOO	mud.crl.com	165.113.1.30	8888
ChibaMOO	chiba.picosof.com	165.227.31.2	7777
Diversity University	erau.db.erau.edu	155.31.1.1	8888
Dragonsfire	moo.eskimo.com	204.122.16.31	7777
DreamMOO	Feenix.metronet.com	192.245.137.1	8888
Entropy	monsoon.weather .brockport.edu	137.21.201.3	7777
Eon	mcmuse.mc.maricopa.edu	140.198.66.28	8888
Final Frontiers MOO	ugly.microserve.net	192.204.120.2	2499
J-H	jhm.ccs.neu.edu	129.10.111.77	1709
LambdaMOO	lambda.xerox.com	192.216.54.2	8888
MediaMOO	purple-crayon.media .mit.edu	18.85.0.48	8888
Metaverse	metaverse.io.com	199.170.88.12	7777
MuMOO	chestnut.enmu.edu	192.94.216.74	7777
OutWorld	tanelorn.king.ac.uk	141.241.224.102	7777
PMC	hero.village .virginia.edu	128.143.200.59	7777
TrekMOO	trekmoo.microserve.com	192.204.120.2	2499
ValhallaMOO	valhalla.acusd.edu	192.55.87.211	4444

IPR2025-00714
DATABRICKS EX1008 Page 384

World MOO	chiba.picosof.com	165.227.31.2	1234
WorldMOO	sequoia.picosof.com	165.227.31.2	1234
* ZenMOO	cheshire.oxy.edu	134.69.1.253	7777

TYPE moo-portuguese (1)

No information available.

MOOsaico	moo.di.uminho.pt	193.136.20.102	7777

TYPE muck (16)

A tiny spinoff chat MUD with an internal programming language similar to FORTH.

AnimeMUCK	tcp.com	128.95.44.29	2035
Brazilian Dreams	red_panda.tbyte.com	198.211.131.13	4201
CaveMUCK	cave.tcp.com	128.95.44.29	2283
Delusions	iglou.com	192.107.41.3	4999
Divination Web	bill.math.uconn.edu	137.99.17.5	9393
FurToonia	lostsouls.desertmoon.com	198.102.68.58	8888
FurryMUCK	sncils.snc.edu	138.74.0.10	8888
HoloMUCK	collatz.mcrcim.mcgill.edu	132.206.78.1	5757
NAILS	flounder.rutgers.edu	128.6.128.5	5150
NonSense MUCK	rigel.dsif.fee.unicamp.br	143.106.8.131	4201
Quartz Paradise	quartz.rutgers.edu	128.6.60.6	9999

IPR2025-00714
DATABRICKS EX1008 Page 385

RealmsMUCK	tcp.com	128.95.44.29	7765
Tapestries MUCK	tcp.com	128.95.44.29	2069
Unbridled Desires	epona.magibox.net	199.171.80.3	8888
V-MUCK	mserv.wizvax.com	199.181.141.3	5201
WizMuck	mserv.wizvax.com	199.181.141.3	4201

TYPE mudwho (2)

Telnet here to get a list of who's on across many different MUDs!

Nova	nova.tat.physik .uni-tuebingen.de	134.2.62.161	6889
okwho	riemann.math .okstate.edu	139.78.112.4	6889

TYPE muse (12)

A tiny mush spinoff chat MUD.

BTech3056	btech.netaxs.com	198.69.186.38	3056
BattleTech	step.polymtl.ca	132.207.7.32	3026
Fantasia	betz.biostr .washington.edu	128.95.44.22	4201
FlatEarth	neumann.cba.csuohio.edu	137.148.20.3	7719
HideawayMUSE	donal.dorsai.org	198.3.127.6	4201
MicroMUSE	CHEZMOTO.AI.MIT.EDU	18.43.0.102	4201
OceanaMUSE	k12.cnidr.org	128.109.179.45	4201
Rhostshyl	RHOSTSHYL.CIT .CORNELL.EDU	128.253.180.15	4201
TOS TrekMuse	siher.Stanford.EDU	36.109.0.64	1701

IPR2025-00714
DATABRICKS EX1008 Page 386

TimeMuse	murren.ai.mit.edu	18.43.0.179	4201
TrekMuse	laurel.cnidr.org	128.109.179.14	1701
VegaMuse II	planck.sos.clarkson.edu	128.153.32.14	2095

TYPE mush (39)

A tiny spinoff chat MUD.

Amarynth	pharos.acusd.edu	192.195.155.201	9999
ApexMUSH	apex.ccs.yorku.ca	130.63.237.12	4201
Belior Rising	brazil-nut.enmu.edu	192.94.216.80	4301
CamelotMUSH	cadman.cit.buffalo.edu	128.205.3.103	5440
Chrome	colossus.acusd.edu	192.195.155.200	4444
City of Darkness	melandra.cs.man.ac.uk	130.88.240.110	2000
Conspiracy!	almond.enmu.edu	192.94.216.77	1066
CryptMUSH	cadman.cit.buffalo.edu	128.205.3.103	7171
CrystalMUSH	moink.nmsu.edu	128.123.8.115	6886
Cybermush	tlaloc.cms.dmu.ac.uk	146.227.103.1	6250
The DAMNED	janus.LIBRARY.CMU.EDU	128.2.21.7	6250
DarkMetal	cheops.acusd.edu	192.195.155.203	4444
DeepSeas	a.cs.okstate.edu	139.78.113.1	6250
DragonDawn	cashew.enmu.edu	192.94.216.78	2222
Dragonlance	yacht.slip.andrew.cmu.edu	128.2.116.75	6250
Dune	mellers1.psych.berkeley.edu	128.32.243.78	4201
Garou	party.apple.com	130.43.2.10	7000

IPR2025-00714
DATABRICKS EX1008 Page 387

Global MUSH	lancelot.cif.rochester.edu	128.151.220.22	4201
Hemlock MUSH	PELYCO.SOAR.GSIA.CMU.EDU	128.2.18.7	1973
ImagECastle	fogey.stanford.edu	36.22.0.31	4201
LegionMUSH	a.cs.okstate.edu	139.78.113.1	2996
Masquerade	phobos.unm.edu	129.24.8.3	4444
NarniaMush	argo.unm.edu	129.24.9.24	6250
NeverEnding Story	jove.cs.pdx.edu	131.252.21.12	9999
PernMUSH	cesium.clock.org	130.43.2.43	4201
PrairieMUSH	firefly.prairienet.org	192.17.3.3	4201
Shards	vesta.unm.edu	129.24.120.253	7777
SouCon	beechnut.enmu.edu	192.94.216.86	4201
Spatial Wastes	chestnut.enmu.edu	192.94.216.74	2001
StarWars	starwars.wis.com	199.3.240.54	4402
SwordsMUSH	world.std.com	192.74.137.5	4201
Texas Twilight	krynn.solace.mh.se	193.10.118.130	6250
# The Awakening	cestus.gb.nrao.edu	192.33.116.67	9999
The Storyteller Circle	draco.unm.edu	129.24.96.4	6666
TinyCWRU	caisr2.caisr.cwru.edu	129.22.24.22	4201
TinyTim	myelin.uchc.edu	155.37.1.251	5440
TooMUSH	cestus.gb.nrao.edu	192.33.116.67	7171
Two Moons Mush	lupine.org	198.4.75.40	4201
XMUSH	xmen.esi.com	192.82.101.106	1994

IPR2025-00714
DATABRICKS EX1008 Page 388

TYPE oxmud (2)

No information available.

Island	teaching4.physics.ox.ac.uk	163.1.245.204	2092
Island	teaching4.physics.ox.ac.uk	163.1.245.204	2093

TYPE silly (1)

No information available.

Chomestoru	dfw.net	198.175.15.10	4000

TYPE talker (7)

A little closer to IRC than a MUD. Definitely a place to hang out and chat.

Addicted	sun1.gwent.ac.uk	193.63.82.1	6666
DS9	rivendel.slip.umd.edu	128.8.11.201	3000
Foothills	marble.bu.edu	128.197.10.75	2010
Gamma Quadrant	rivendel.slip.umd.edu	128.8.11.201	6000
MBA4	jumper.mcc.ac.uk	130.88.202.26	3214
* SomeWhere Else	kehleyr.phys.ttu.edu	129.118.41.9	2010
Surfers	muscle.rai.kcl.ac.uk	137.73.16.2	3232

TYPE teeny (2)

A tiny lookalike chat MUD.

(EVIL!)Mud	intac.com	198.6.114.2	4201
ApexMUD	apex.yorku.ca	130.63.237.12	4201

IPR2025-00714
DATABRICKS EX1008 Page 389

TYPE tiny (1)

The original chat MUD.

DragonMUD	satan.ucsd.edu	132.239.1.7	4201

TYPE uber (1)

No information available.

The Land of Drogon	drogon.meiko.com	192.131.107.11	6123

Key

? No successful connect on record—may be removed from future lists.

* Last successful connect more than 7 days ago.

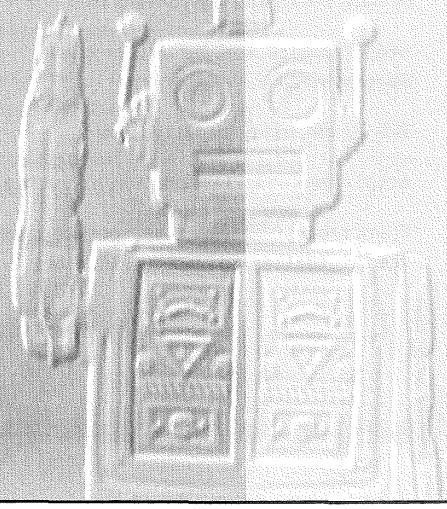
\# Last successful connect more than 3 days ago.

IPR2025-00714
DATABRICKS EX1008 Page 390

appendix G

List of World Wide Web Spiders and Robots

The following list of World Wide Web Robots, Wanderers, and Spiders is compiled from Martijn Koster's List of Active Robots found at `http://web.nexor.co.uk/mak/doc/robots/active.html`.



IPR2025-00714
DATABRICKS EX1008 Page 391

The JumpStation Robot

Home Page: `http://www.stir.ac.uk/jsbin/js`

Run by: Jonathon Fletcher, `J.Fletcher@stirling.ac.uk`

From: `pentland.stir.ac.uk`

User-Agent: JumpStation

Its purpose is to generate a Resource Discovery database. The Proposed Standard for Robot Exclusion is supported. It is a set of stand-alone programs written in Perl 4, C, and C++. Verion I has been in development since September 1993, and has been running on several occasions; the last run was between February 8th and February 21st, 1995. Version II is under development.

RBSE Spider

Home Page: `http://rbse.jsc.nasa.gov/eichmann/rbse.html`

Run by: Dr. David Eichmann, `eichmann@rbse.jsc.nasa.gov`

From: `rbse.jsc.nasa.gov` (192.88.42.10)

User-Agent: RBSE-Spider/0,1a

Seems to retrieve documents more than once. Its purpose is to generate a Resource Discovery database, and generate statistics. It is a stand-alone program written in C, and Oracle, WAIS. Developed and operated as part of the NASA-funded Repository Based Software Engineering Program at the Research Institute for Computing and Information Systems, University of Houston—Clear Lake.

The WebCrawler

Home Page: `http://webcrawler.com/`

Run by: Brian Pinkerton, `bp@biotech.washington.edu`

`http://www.cs.washington.edu/homes/bp/bp.html`

From: `surfski.webcrawler.com`

User-Agent: WebCrawler/2.0 libwww/3.0

Its purpose is to generate a Resource Discovery database, and generate statistics. The WebCrawler does a breadth-first walk, and indexes content as well as URLs, and so on. The Proposed Standard for Robot Exclusion is not supported. It is a stand-alone program written in C. The WebCrawler originated as an experiment in Internet resource discovery at the University of Washington in 1994. Today, it is operated by America Online as a service to the Internet. robots.txt support is coming soon!

The NorthStar Robot

Home Page: `http://comics.scs.unr.edu:7000/top.html`

Run by: Fred Barrie, `barrie@unr.edu`

`http://comics.scs.unr.edu/people/barrie.html`

Billy Barron, `billy@utdallas.edu`

`http://www.utdallas.edu/acc/billy.html`

From: `frognot.utdallas.edu`, possibly other sites in `utdallas.edu` and in `cnidir.org`.

IPR2025-00714
DATABRICKS EX1008 Page 392

User-Agent: NorthStar

More information, including a search interface, is available on the NorthStar Database. Recent runs (April 26, 1994) will concentrate on textual analysis of the Web versus GopherSpace (from the Veronica data) as well as indexing.

W4 (World Wide Web Wanderer)

Run by: Matthew Gray, mkgray@mit.edu

User-Agent: WWWWanderer v3.0 by Matthew Gray, mkgray@mit.edu

Run initially in June 1993, its aim is to measure the growth in the Web.

The Fish Search

Home Page: http://www.win.tue.nl/help/help-on-fish-all.html

Written by: Paul De Bra, debra@win.tue.nl

http://www.win.tue.nl/win/cs/is/debra/

Software: file://ftp.win.tue.nl/pub/infosystems/www/

Paper: http://www.win.tue.nl/win/cs/is/reinpost/www94/

From: Not set, but it's usually run from www.win.tue.nl.

User-Agent: Fish-Search-Robot

Its purpose is to discover resources on the fly. The Proposed Standard for Robot Exclusion is not supported because of the incurred overhead. It is a stand-alone program written in C, but a version exists that is integrated into the Tubingen Mosaic 2.4.2 browser (also written in C). Originated as an addition to Mosaic for X. Available as a stand-alone program from ftp://ftp.win.tue.nl/pub/infosystems/www/fish-search.tar.gz.

The Python Robot

Home Page: http://info.cern.ch/hypertext/WWW/Tools/Python/Overview.html

Written by: Guido van Rossum, Guido.van.Rossum@cwi.nl

Software: ftp://ftp.cwi.nl/pub/python/python0.9.8.tar.Z

ftp://ftp.cwi.nl/pub/python/demo/www/

HTML Analyzer

Run by: James E. Pitkow, pitkow@aries.colorado.edu

Its aim is to check validity of Web servers. Not sure if it has ever been run remotely.

MOMspider

Home Page: http://www.ics.uci.edu/WebSoft/MOMspider

IPR2025-00714
DATABRICKS EX1008 Page 393

Written by: Roy Fielding, fielding@ics.uci.edu

Software: ftp://liege.ics.uci.edu/pub/arcadia/MOMspider/MOMspider-1.00.tar.Z

Paper: http://www.ics.uci.edu/WebSoft/MOMspider/WWW94/paper.html

From: Can run from anywhere.

User-Agent: MOMspider/1.00 libwww-perl/0.40

MOMspider is a Web-roaming robot that specializes in the maintenance of distributed hypertext infostructures. Its purpose is to validate links, and generate statistics. The program is stand-alone and written in Perl. The Proposed Standard for Robot Exclusion is supported. Originated as a research project at the University of California, Irvine, in 1993. Presented at the First International WWW Conference in Geneva, 1994.

HTMLgobble

Run by: Andreas Ley, ley@rz.uni-karlsruhe.de

From: tp70.rz.uni-karlsruhe.de

User-Agent: HTMLgobble v2.2

Software: ftp://ftp.rz.uni-karlsruhe.de/pub/net/www/tools/htmlgobble.tar.gz

A mirroring robot. Configured to stay within a directory, sleeps between requests, and the next version will use HEAD to check whether the entire document needs to be retrieved.

WWWW—the WORLD WIDE WEB WORM

Home Page: http://www.cs.colorado.edu/home/mcbryan/WWWW.html

http://www.cs.colorado.edu/home/mcbryan/WWWWintro.html

Run by: Oliver McBryan, mcbryan@piper.cs.colorado.edu

http://www.cs.colorado.edu/home/mcbryan/Home.html

Paper: http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps

From: piper.cs.colorado.edu

Another indexing robot that has quite flexible search options.

WM32 Robot

Home Page: http://www-ihm.lri.fr/~tronche/W3M2/

Written by: Christophe Tronche, Christophe.Tronche@lri.fr

http://www-ihm.lri.fr/~tronche/

From: E-mail address of the operator, usually tronche@lri.fr.

User-Agent: W3M2/x.xx

Its purpose is to generate a Resource Discovery database, validate links, validate HTML, and generate statistics. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in Perl 4, Perl 5, and C++.

IPR2025-00714
DATABRICKS EX1008 Page 394

Websnarf

Run by: Charlie Stross, charless@sco.com

From: ruddles.london.sco.com

A WWW mirror designed for offline browsing of sections of the Web. No longer running.

The Webfoot Robot

Run by: Lee McLoughlin, L.McLoughlin@doc.ic.ac.uk

http://web.doc.ic.ac.uk/f?/lmjm

From: phoenix.doc.ic.ac.uk

First spotted in Mid-February 1994. Further information unavailable.

Lycos

Home Page: http://lycos.cs.cmu.edu/

Run by: Dr. Michael L. Mauldin, fuzzy@cmu.edu

http://fuzine.mt.cs.cmu.edu/mlm/home.html

From: fuzine.mt.cs.cmu.edu

User-Agent: Lycos/x.x

This is a research program in providing information retrieval and discovery in the Web, using a finite memory model of the Web to guide intelligent, directed searches for specific information needs. Lycos also complies with the latest robot exclusion standard.

ASpider (Associative Spider)

Written by: Fred Johansen, fred@nvg.unit.no

http://www.nvg.unit.no/~fred/

From: fredj@nova.pvv.unit.no

User-Agent: ASpider/0.09

Currently under construction, this spider is a CGI script that searches the Web for keywords given by the user through a form.

SG-Scout

Home Page: http://www-swiss.ai.mit.edu/~ptbb/SG-Scout/SG-Scout.html

Run by: Peter Beebee, ptbb@ai.mit.edubeebee@parc.xerox.com

User-Agent: SG-Scout

From: Set to operator, usually from beta.xerox.com

The SG-Scout robot was developed for the Xerox Palo Alto Research Center as part of a project involving the development of a new form of directed Web browser. The SG-Scout is compliant with the Standard for Robot Exclusion. The SG-Scout was run in late June and again in early August 1994. In August the robot discovered over 7,250 Web servers and 250,000 html and text pages available through the HTTP protocol. It performs a "server-oriented" breadth-first search in a round-robin fashion, with multiple processes.

IPR2025-00714
DATABRICKS EX1008 Page 395

EIT Link Verifier Robot

Home Page: `http://wsk.eit.com/wsk/dist/doc/admin/webtest/verify_links.html`

Written by: Jim McGuire, `mcguire@eit.COM`

Software: `ftp://ftp.eit.com/pub/wsk/doc/README.verify_links`

From: Can be run by anyone from anywhere.

User-Agent: EIT-Link-Verifier-Robot/0.2

Announced on July 12, 1994. Combination of an HTML form and a CGI script that verifies links from a given starting point (with some controls to prevent it going off-site or limitless).

NHSE Web Forager

Run by: Bob Olson, `olson@mcs.anl.gov`

From: Usually `*.mcs.anl.gov`

User-Agent: NHSEWalker/3.0

Its purpose is to generate a Resource Discovery database. This robot is gathering data to do a full-text glimpse and provide a Web interface for it. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in Perl 5.

WebLinker

Home Page: `http://www.cern.ch/WebLinker/`

Written by: James Casey, `casey@ptsun00.cern.ch`

`http://www.maths.tcd.ie/hyplan/jcasey/jcasey.html`

Paper: `http://www.cern.ch/WebLinker/Paper/Welcome.html`

User-Agent: WebLinker/0.0 libwww-perl/0.1

It is a tool which traverses a section of Web, doing URN-URL conversion. It will be used as a post-processing tool on documents created by automatic converters such as LaTeX2HTML or WebMaker.

At the moment it works at full speed, but is restricted to local sites. External GETs will be added, but these will be running slowly.

Emacs W3 Search Engine

Home Page: `http://www.cs.indiana.edu/elisp/w3/w3_1.html`

`http://www.cs.indiana.edu/elisp/w3/w3_toc.html#SEC39`

Written by: William M. Perry, `wmperry@spry.com`

`http://www.cs.indiana.edu/hyplan/wmperry.html`

From: Various machines

User-Agent: Emacs-w3/v*.*

Its purpose is to generate a Resource Discovery database. This is part of the w3 browser mode for Emacs, and is written in Lisp. The Proposed Standard for Robot Exclusion is not supported.

IPR2025-00714
DATABRICKS EX1008 Page 396

Arachnophilia

Run by: Vince Taluskie, taluskie@utpapa.ph.utexas.edu

http://www.ph.utexas.edu/people/vince.html

From: halsoft.com

User-Agent: Arachnophilia

The purpose (undertaken by HaL Software) of this run is to collect approximately 10,000 html documents for testing automatic abstract generation. This program honors the robot exclusion standard and waits one minute between requests to a given server.

Mac WWWWorm

Written by: Sebastien Lemieux, lemieuse@ERE.UMontreal.CA

http://alize.ere.umontreal.ca:8001/~lemieuse/sebast.html

This is a French Keyword-searching robot for the Mac, written in HyperCard. The author has decided not to release this robot to the public.

ChURL

Home Page: http://www.engin.umich.edu/~yunke/scripts/#Churl

Run by: Justin Yunke, yunke@umich.edu

http://www.engin.umich.edu/~yunke/

Software: http://www.engin.umich.edu:80/~yunke/feedback/

A URL checking robot, which stays within one step of the local server. However, the remote links can be checked for a connection. It is written in Perl.

Tarspider

Home Page: http://www.chemie.fu-berlin.de/user/chakl/Spider.html

Run by: Olaf Schreck, chakl@fu-berlin.de

http://www.inf.fu-berlin.de/~weisshuh/chakl/ChaklHome.html

From: chakl@fu-berlin.de

User-Agent: tarspider version

A mirroring robot.

The Peregrinator

Home Page: http://www.maths.usyd.edu.au:8000/jimr/pe/Peregrinator.html

Run by: Jim Richardson, jimr@maths.su.oz.au

http://www.maths.usyd.edu.au:8000/jimr.html

User-Agent: Peregrinator-Mathematics/0.7

This robot, in Perl V4, commenced operation in August 1994 and is being used to generate an index called MathSearch of documents on Web sites connected with mathematics and statistics. It ignores off-site links, so does not stray from a list of

IPR2025-00714
DATABRICKS EX1008 Page 397

servers specified initially. The robot follows the exclusion standard and accesses any given server no more than once every several minutes.

Checkbot

Written by: Dimitri Tischenko, D.B.Tischenko@TWI.TUDelft.NL

http://www.twi.tudelft.nl/People/D.B.Tischenko.html

Run by: Hans de Graaff, j.j.degraaff@twi.tudelft.nl

From: Usually dutifp.twi.tudelft.nl

User-Agent: checkbot.pl/x.x libwww-perl/x.x

The Proposed Standard for Robot Exclusion is not supported. It is a stand-alone program written in Perl 5.

Webwalk

Written by: Rich Testardi, rpt@fc.hp.com

User-Agent: webwalk

Its purpose is to generate a Resource Discovery database, validate links, validate HTML, perform mirroring, copy document trees, and generate statistics. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in C. Webwalk is easily extensible to perform virtually any maintenance function which involves web traversal, in a way much like the -exec option of the find(1) command. Webwalk is usually used behind the HP firewall.

Harvest

Home Page: http://harvest.cs.colorado.edu/

Run by: Darren Hardy, hardy@bruno.cs.colorado.edu

Software: http://harvest.cs.colorado.edu/harvest/gettingsoftware.html

Papers: http://harvest.cs.colorado.edu/harvest/papers.html

From: bruno.cs.colorado.edu

A Resource Discovery Robot, part of the Harvest Project. Pauses 1 second between requests (by default). Note that Harvest's motivation is to index community- or topic-specific collections, rather than to locate and index all HTML objects that can be found. Also, Harvest enables users to control the enumeration several ways, including stop lists and depth and count limits. Therefore, Harvest provides a much more controlled way of indexing the Web than is typical of robots.

Katipo

Home Page: http://www.vuw.ac.nz/~newbery/Katipo.html

Run by: Michael Newbery, Michael.Newbery@vuw.ac.nz

http://www.vuw.ac.nz/~newbery

From: Michael.Newbery@vuw.ac.nz

User-Agent: Katipo/1.0

The Proposed Standard for Robot Exclusion is not supported. It is a stand-alone program written in C.

IPR2025-00714
DATABRICKS EX1008 Page 398

A Macintosh robot that periodically (typically, once per day) walks through the global history files provided by some browsers (Mosaic, Netscape), looking for pages that have changed since last visited. It emits only HEAD queries and prefers to work through proxy (caching) servers.

InfoSeek Robot

Written by: Steve Kirsch, `stk@infoseek.com`

From: `corp-gw.infoseek.com`

User-Agent: InfoSeek Robot 1.0

Its purpose is to collect information to use in a "WWW Pages" database in InfoSeek's information retrieval service. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in Python.

Collects WWW pages for both InfoSeek's free WWW search and commercial search. Uses a unique proprietary algorithm to identify the most popular and interesting WWW pages. Very fast, but never has more than one request per site outstanding at any given time. Has been under refinement for more than a year.

GetURL

Written by: James Burton, `burton@cs.latrobe.edu.au`

`http://www.cs.latrobe.edu.au/~burton/`

Software: `http://www.cs.latrobe.edu.au/~burton/Public/`

User-Agent: GetUrl.rexx v1.0

Its purpose is to validate links, perform mirroring, and copy document trees. The Proposed Standard for Robot Exclusion is not supported. It is a robot written in ARexx (Amiga Rexx), which downloads a hierarchy of documents with a breadth-first search.

Designed as a tool for retrieving Web pages in batch mode without the encumbrance of a browser. Can be used to describe a set of pages to fetch and to maintain an archive or mirror. Is not run by a central site and accessed by clients—is run by the end user or archive maintainer.

Open Text Corporation Robot

Run by: Tim Bray, `tbray@opentext.com`

User-Agent: OMW/0.1 libwww/217

Follows robot exclusion rules and shouldn't visit any host more than once in five minutes.

NIKOS

Home Page: `http://www.rns.com/cgi-bin/nomad`

Written by: Rockwell Network Systems

This is an Internet resource discovery written by Rockwell Network Systems that searches only through document titles.

IPR2025-00714
DATABRICKS EX1008 Page 399

The TkWWW Robot

Home Page: `http://fang.cs.sunyit.edu/Robots/tkwww.html`

Written by: Scott Spetka, `scott@cs.sunyit.edu`

`http://fang.cs.sunyit.edu/Robots/spetka.html`

Software: `http://fang.cs.sunyit.edu/Spetka/tkwww`

TkWWW robots are dispatched from the TkWWW browser their results can be used to guide further browsing. It is designed to search Web neighborhoods to find pages that may be logically related. Robots can also be run in the background to build HTML indexes, compile WWW statistics, collect a portfolio of pictures, or perform any other function that can be described by the TkWWW Tcl extensions.

A Tcl W3 Robot

Home Page: `http://hplyot.obspm.fr/~dl/robo.html`

Written by: Laurent Demailly, `dl@hplyot.obspm.fr`

Software: `http://hplyot.obspm.fr/~dl/geturl.tcl`

`http://hplyot.obspm.fr/~dl/w3cli.tcl`

From: Usually `hplyot.obspm.fr`

User-Agent: dlw3robot/x.y

Its purpose is to validate links, and generate statistics. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in TCL.

TITAN

Written by: Yoshihiko Hayashi, `hayashi@nttnly.isl.ntt.jp`

From: Usually `nttnly.isl.ntt.jp`

User-Agent: TITAN/0.1

Its purpose is to generate a Resource Discovery database, and copy document trees. A primary goal is to develop an advanced method for indexing the WWW documents. By using libwww-perl, the Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in Perl 4.

CS-HKUST WWW Index Server

Home Page: `http://dbx.cs.ust.hk:8000/`

Written by: Budi Yuwono, `yuwono-b@cs.ust.hk`

From: Usually `dbx.cs.ust.hk`

User-Agent: CS-HKUST-IndexServer/1.0

Its purpose is to generate a Resource Discovery database, and validate HTML. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in C. Part of an ongoing research project on Internet Resource Discovery at Department of Computer Science, Hong Kong University of Science and Technology (CS-HKUST).

Spry Wizard Robot

Home Page: `http://www.compuserve.com/wizard/wizard.html`

IPR2025-00714
DATABRICKS EX1008 Page 400

Written by: Spry info@spry.com

Its purpose is to generate a Resource Discovery database. Unfortunately neither the User-agent nor the From HTTP fields are set. It's usually run from wizard.spry.com or tiger.spry.com. Spry is refusing to give any comments about this robot.

Weblayers

Home Page: http://www.univ-paris8.fr/~loic/weblayers/

Written by: Loic Dachary, loic@afp.com

Software: ftp://www.univ-paris8.fr/~loic/weblayers/weblayers-0.0.tar.gz

User-Agent: weblayers/0.0

Its purpose is to validate, cache, and maintain links. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in Perl 5. It is designed to maintain the cache generated by the emacs w3 mode (Netscape replacement) and to support annotated documents (keep them in sync with the original document via diff/patch).

WebCopy

Home Page: http://www.inf.utfsm.cl/~vparada/webcopy.html

Written by: Victor Parada, vparada@inf.utfsm.cl

Software: ftp://ftp.inf.utfsm.cl/pub/utfsm/perl/webcopy.tgz

From: Not set.

User-Agent: WebCopy/(version)

Its purpose is to perform mirroring. The Proposed Standard for Robot Exclusion is not supported. It is a stand-alone program written in Perl 4 or 5. WebCopy can retrieve files recursively using HTTP protocol. It can be used as a delayed browser or as a mirroring tool. It cannot jump from one site to another. It can be used by anyone from anywhere.

Scooter

Home Page: http://scooter.pa-x.dec.com/

Written by: Louis Monier, monier@pa.dec.com

From: Usually scooter.pa-x.dec.com

User-Agent: Scooter/1.0

Its purpose is to generate a Resource Discovery database, and generate statistics. After fetching a page, Scooter will not contact the same host for 100 times the duration of the last fetch. The Proposed Standard for Robot Exclusion is supported. It is a stand-alone program written in C.

Aretha

Written by: Dave Weiner, davew@well.com

A crude robot built on top of Netscape and Userland Frontier, a scripting system for Macs.

WebWatch

Home Page: http://www.xensei.com/users/janos/specter/

IPR2025-00714
DATABRICKS EX1008 Page 401

Written by: Joseph Janos, janos@specter.com

From: Not set.

User-Agent: WebWatch

Its purpose is to validate HTML, and generate statistics. The Proposed Standard for Robot Exclusion is not supported. It is a stand-alone program written in C++. Checks URLs modified since a given date. Shareware.

IPR2025-00714
DATABRICKS EX1008 Page 402

Chapter 1

(Apple 1988) Apple Computer Inc. "Project 2000—A Knowledge Navigator." Videotape. Available from Apple Video Fulfillment Program, 1-800-627-0230, Mar 8, 1988.

(Bates 1994) Bates, Joseph. "The role of emotion in believable agents." *Commun. ACM*, 37, 7, pp.122–125, 1994.

(BDDWW 1993) Birmingham, W., T. Darr, E. Durfee, A. Ward, and M. Wellman. "Supporting mechatronic design via a distributed network of intelligent agents." In *AI in Collaborative Design, workshop notes*, J.S. Gero and M.L. Maher, Eds. University of Sydney, 1993.

(BFKM 1985) Brownston, L., R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming.* Reading, Mass., Addison-Wesley, 1985.

(Bledsoe 1986) Bledsoe, Woody. "I had a dream: AAAI presidential address." *AI Magazine*, vol. 7, no. 1, pp.57–61, 1986.

(BLR 1992) Bates, J., A.B. Loyall, and W.S. Reilly. "An architecture for action, emotion, and social behavior." In *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World.* S. Martino al Cimino, Italy, 1992.

(Boden 1991) Boden, M.A. *The Creative Mind: Myths and Mechanisms. (Expanded Edition).* Basic Books, New York; Abacus, London, 1991.

IPR2025-00714
DATABRICKS EX1008 Page 403

(Boden 1994) Boden, M.A. "Agents and creativity." *Commun. ACM.* 37, 7, pp.117–121, 1994.

(Brooks 1986) Brooks, Rodney. "A robust layered control system for a mobile robot." IEEE J. Robotics and Automation, RA-2, pp.14–23, 1986.

(Brooks 1989) Brooks, R. "A robot that walks: Emergent behavior from a carefully evolved network." *Neural Computation*, 1, 2, (Summer 1989), pp.253–262.

(Brooks 1990) Brooks, R. "Elephants don't play chess." In *Designing Autonomous Agents*, Pattie Maes Ed. pp.3–15, MIT Press, Cambridge, Mass., 1990.

(Brooks 1991) Brooks, R. "Intelligence without representation." *AI Journal*, 47, 1, 1991.

(Cheong 1992a) Cheong, F.C. "Oasis: An Agent-Oriented Programming Language for Heterogeneous Distributed Environment." Ph.D. Thesis. University of Michigan, 1992.

(Cheong 1992b) Cheong, F.C. `ftp://ftp.eecs.umich.edu/software/oasis/`.

(Cutkosky 1993) Cutkosky, M. et al. "PACT: An experiment in integrated engineering systems." *Computer* 26, 1, pp.28–37, 1993.

(Cypher 1993) Cypher, A., Ed. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, Mass., 1993.

(DB 1992) Darr, T.P. and W.P. Birmingham. *Concurrent Engineering: An Automated Design Space Approach.* Tech. Rep. CSE-TR-149-92, University of Michigan, 1992.

(DBMMZ 1992) Dent, L., J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. "A personal learning apprentice." In *Proceedings of the International Joint Conference on Artificial Intelligence.* 1992.

(DS 1983) Davis, R. and R.G. Smith. "Negotiation as a metaphor for distributed problem-solving." *Artificial Intelligence.* 20, 1, pp.63–109, 1983.

(ECJS 1994) Edmonds, E.A., L. Candy, R. Jones, and B. Soufi. "Support for collaborative design: agents and emergence." *Commun. ACM.* 37, 7, pp.41–47, 1994.

(EW 1994) Etzioni, O. and D. Weld. "A softbot-based interface to the Internet." *Commun. ACM.* 37, 7, pp.72–76, 1994.

(GK 1994) Genesereth, M. and S. Ketchpel. "Software Agents." *Commun. ACM*, 37, 7, pp.48–53, 1994.

(GL 1994) Guha, R.V. and D.B. Lenat. "Enabling agents to work together." *Commun. ACM.* 37, 7, pp.126–142, 1994.

(GM 1994) General Magic Inc. *The World of Telescript Technology.* 1994.

(Greif 1994) Greif, I. "Desktop agents in group-enabled products." *Commun. ACM.* 37, 7, pp.100–105, 1994.

(Gruber 1991) Gruber, T. "Ontolingua: A mechanism to support portable ontologies." KSL-91-66, Stanford Knowledge Systems Laboratory, 1991.

(HF 1992) Hersey, G. and R. Freedman. *Possible Palladian Villas (Plus a Few Instructively Impossible Ones).* MIT Press, Cambridge, Mass., 1992.

IPR2025-00714
DATABRICKS EX1008 Page 404

(Hodgson 1990) Hodgson, P. "Understanding Computing, Cognition, and Creativity." M.Sc. thesis, University of the West of England, 1990.

(HP 1989) Hewlett-Packard. "HP New Wave Agent Guide." Santa Clara, Calif., Oct 1989.

(JDMMZ 1991) Jourdan, J., L. Dent, J. McDermott, T. Mitchell, and D. Zabowski. "Interfaces that learn: A learning apprentice for calendar management." *Tech. Rep.* CMU-CS-91-135, Carnegie Mellon Univ., Pittsburgh, PA., 1991.

(Kay 1984) Kay, A. "Computer software." *Scientific American.* 251, 3, pp.53–59, 1984.

(Kay 1990) Kay, A. "User Interface: A personal view." In *The Art of Human-Computer Interface Design.* B. Laurel ed. Addison-Wesley, Reading, Mass., 1990.

(KE 1981) Koning, H. and J. Eizenberg. "The language of the prairie: Frank Lloyd Wright's prairie houses." *Environmental Plan.* B 8, 1981, pp.295–323.

(KSC 1994) Kautz, H., B. Selman, and M. Coen "Bottom-up design of software agents." *Commun. ACM.* 37, 7, pp.143–146, 1994.

(LB 1983) Loyall, A.B. and J. Bates. "Real-time control of animated broad agents." In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society,* Boulder, Colorado, 1993.

(Lenat 1983) Lenat, D.B. "The role of heuristics in learning by discovery: Three case studies." In *Machine Learning: An Artificial Intelligence Approach.* R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds. Tioga, Palo Alto, Calif., 1983.

(LGPPS 1990) Lenat, D.B., R.V. Guha, K. Pittman, D. Pratt, and M. Shepherd. "Cyc: Toward programs with common sense." *Commun. ACM.* 33, 8, pp.30–49, 1990.

(LNR 1987) Laird, J., A. Newell, and P. Rosenbloom. "SOAR: An Architecture for general intelligence." *Artificial Intelligence.* J. 33, 1, pp.1–64, 1987.

(Maes 1989) Maes, Pattie. "How to do the right thing." *Connect. Sci.* 1, 3, pp.291–323, 1989.

(Maes 1994) Maes, Pattie. "Agents that reduce work and information overload." *Commun. ACM.* 37, 7, pp.30–40, 1994.

(MCFMZ 1994) Mitchell, T., R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. "Experience with a learning personal assistant." *Commun. ACM.* 37, 7, pp.80–91, 1994.

(MGLRR 1987) Malone, T.W., K.R. Grant, K-Y. Lai, R. Rao, and D. Rosenblitt. "Semi-structured messages are surprisingly useful for computer-supported coordination." *ACM Trans. Information Systems.* 5, 2 (April, 1987), pp.115–131.

(Minsky 1985) Minsky, M. *The Society of Mind.* Simon and Schuster, New York, 1985.

(MWFF 1992) Medina-Mora, R., T. Winograd, R. Flores, and F. Flores. "The action workflow approach to workflow management technology." In *Proceedings of the '92 Computer-Supported Cooperative Work '92 Conference.* pp.281–288, 1992.

(Negroponte 1970) Negroponte, N. *The Architecture Machine: Towards a More Human Environment.* MIT Press, Cambridge, Mass., 1970.

IPR2025-00714
DATABRICKS EX1008 Page 405

(Negroponte 1989) Negroponte, N. "An iconoclastic view beyond the desktop metaphor." *Journal on Human-Computer Interaction*, 1, 1, pp.109–113, 1989.

(NFFGPSS 1991) Neches, R., R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout. "Enabling technology for knowledge sharing." *AI Magazine*. 12, 3, pp.36–56, 1991.

(Norman 1994) Norman, D.A. "How might people interact with agents?" *Commun. ACM*. 37, 7, pp.68–71, 1994.

(OCC 1988) Ortony, A., G. Clore, and A. Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1988.

(Pottmyer 1994) Pottmyer, J. "Renegade intelligent agents." SIGNIDR V, Proce. Special Interest Group on Networked Information Discovery and Retrieval, McLean, VA, August 4, 1994. Presentation slides available as `http://www.wais.com/SIGNIDR/Proceedings/SA3/`.

(PYFLHCM 1992) Palaniappan, M., N. Yankelovich, G. Fitzmaurice, A. Loomis, B. Haan, J. Coombs, and N. Meyrowitz. "The envoy framework: An open architecuture for agents." *ACM Trans. Information Systems*. 10, 3, July 1992, pp.233–264.

(Quinlan 1986) Quinlan, J.R. "Induction of decision trees." *Machine Learn*. 1, 1, pp.81–106, 1986.

(Riecken 1994) Riecken, D. "M: An architecture of integrated agents." *Commun. ACM*. 37, 7, pp.106–116, 1994.

(Schneiderman 1988) Scheneiderman, B. "Direct manipulation: A step beyond programming languages." *IEEE Comput*. 16, 8, pp.57–69, 1988.

(SCS 1994) Smith, D., A. Cypher, and J. Spohrer. "KidSim: Programming agents without a programming language." *Commun. ACM*. 37, 7, pp.54–67, 1994.

(Selker 1994) Selker, Ted. "COACH: A teaching agent that learns." *Commun. ACM*. 37, 7, pp.92–99, 1994.

(Smith 1977) Smith, D.C. *Pygmalion, A Computer Program to Model and Stimulate Creative Thought*. Birkhauser Verlag, Basel, Switzerland, 1977.

(TJ 1981) Thomas, F. and O. Johnston. *Disney Animation: The Illusion of Life*. Abbeville Press, New York, 1981.

(Waugh 1992) Waugh, I. "Improvisor." *Music Tech*. (Sept. 1992), pp.70–73.

(White 1994) James E. White. "Telescript Technology: The Foundation for the Electronic Marketplace." *General Magic White Paper*. 1994.

Chapter 2

(Cerf 1992) Cerf, Vinton. "A Brief History of the Internet and Related Networks," Internet Society, Reston, Virginia, 1992.

(Chapin 1992) Chapin, Lyman (Chair). "The Internet Standards Process," RFC 1310, Internet Activities Board, March 1992.

(Denning 1989) Denning, Peter. "The ARPANET after twenty years." *American Scientist*. Nov-Dec, 1989, pp.530–534.

IPR2025-00714
DATABRICKS EX1008 Page 406

(Horton 1983) Horton, M.R. Standard for interchange of Usenet messages. RFC 850. SRI International, Menlo Park, California, June 1983.

(KL 1986) B. Kantor and P. Lapsley. "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News." RFC 977, UC San Diego, UC Berkeley, February 1986.

(Lottor 1992) Lottor, Mark. RFC 1296: Internet Growth (1981-91). 1992.

(Merit 1992) Merit Network, Inc. "NSFNET: Bringing the World of Ideas Together." April, 1992. `ftp://nis.nsf.net/nsfnet/nsfnet.overview`.

(Moore 1994a) Moore, Martin. "Introducing the Internet." From *The Internet Unleashed*. Sams.net Publishing, Indianapolis, Indiana. 1994.

(Moore 1994b) Moore, Martin. "The Internet Today and Tomorrow." From *The Internet Unleashed*. Sams.net Publishing, Indianapolis, Indiana. 1994.

(Moore 1994c) Moore, Martin. "The Future of the Internet." From *The Internet Unleashed*. Sams.net Publishing, Indianapolis, Indiana. 1994.

(Quarterman 1993) Quarterman, John S. *Internet Growth*. Matrix News, 3(12), December 1993. `gopher://gopher.tic.com/00/matrix/news/v3/inetgrow.312`.

(QH 1986) Quarterman, John S. and Josiah C. Hoskins. "Notable Computer Networks." *Commun. ACM*, vol.29, no.10, October 1986, pp.932–971.

(QCM 1994) Quarterman, John S. and Carl-Mitchell, Smoot. *The Internet Connection: System Connectivity and Configuration*. Addison-Wesley, Reading, Massachusetts, 1994.

(Tanenbaum 1981a) Tanenbaum, Andrew. "Network protocols." *ACM Computing Survey*, vol.13, no.4, pp.453–489, Dec, 1981.

(Tanenbaum 1981b) Tanenbaum, Andrew. *Computer Networks*. Prentice-Hall Software Series, Prentice Hall, Englewood Cliffs, New Jersey, 1981.

Chapter 3

(AMLJTA 1993) Anklesaria, F., M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. "The Internet Gopher Protocol: A Distributed Document Search and Retrieval Protocol." RFC 1436, University of Minnesota, March 1993.

(BL 1994) Bernes-Lee, T. "Universal Resource Identifiers in WWW" RFC 1630, CERN. June 1994.

(BLCLNS 1994) Bernes-Lee, T., R. Cailliau, A. Luotonen, H. Nielsen, and A. Secret. "The World Wide Web." *Commun. ACM*. 37, 8, pp.76–82, 1994.

(BLC 1995) Berners-Lee, T. and D. Connolly. "HyperText Markup Language—2.0." Work in Progress. June 1995. `ftp://ds.internic.net/internet-drafts/draft-ietf-html-spec-04.txt`.

(BLFN 1995) Bernes-Lee, T., R. T. Fielding, and H. F. Nielsen. "Hypertext Transfer Protocol—HTTP/1.0." *Work in Progress*. March 1995. `ftp://ds.internic.net/internet-drafts/draft-ietf-http-v10-spec-00.txt`.

(BLMM 1994) Berners-Lee, T., L. Masinter, and M. McCahill. "Uniform Resource Locators (URL)." RFC 1738, CERN, Xerox PARC, University of Minnesota, October 1994.

IPR2025-00714
DATABRICKS EX1008 Page 407

(BF 1993) Borenstein, N. and N. Freed. "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies." RFC 1521, Bellcore, Innosoft, September 1993.

(DKMSSWSG 1990) Davis, F., B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification." (v1.5), Thinking Machines Corporation, April 1990.

(DR 1995) December, John and Neil Randall. *The World Wide Web Unleashed.* Sams.net Publishing, Indianapolis, Indiana. 1995.

(Goldfarb 1990) Goldfarb, Charles. *The SGML Handbook.* Clarendon Press, Oxford, 1990.

(Raggett 1995) Raggett, Dave. "HyperText Markup Language Specification Version 3.0." *Work in Progress.* W3C. March 1995. `ftp://ds.internic.net/internet-drafts/draft-ietf-html-specv3-00.txt`.

Chapter 4

(BDHMS 1994) Bowman, C. Mic, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. "The Harvest Information Discovery and Access System." Proceedings of the Second International World Wide Web Conference, pp. 763–771, Chicago, Illinois, October 1994.

(DR 1995) December, John and Neil Randall. *The World Wide Web Unleashed.* Sams.net Publishing, Indianapolis, Indiana. 1995.

(December 1994) December, John. "New Spiders Roam the Web." *Computer-Edited Communication Magazine.* 1(5), Sep. 1, 1994. `http://www.rpi.edu/~decemj/cmc/mag/1994/sep/spiders.html`.

(DP 1994) DeBra, P. and R. Post. "Information Retrieval in the World Wide Web: Making Client-based searching feasible." Proceedings of the First International World Wide Web Conference. R. Cailliau, O. Nierstrasz, M. Ruggier (eds.), Geneva, 1994.

(Eichmann 1994) Eichmann, David. "The RBSE Spider—Balancing Effective Search Against Web Load." Proceedings of the First International World Wide Web Conference. R. Cailliau, O. Nierstrasz, M. Ruggier (eds.), Geneva, 1994.

(Koster 1994) Koster, Martijn. "ALIWEB: Archie-Like Indexing in the Web." Proceedings of the First International World Wide Web Conference. R. Cailliau, O. Nierstrasz, M. Ruggier (eds.), Geneva, 1994.

(Koster 1994a) Koster, Martijn. "A standard for robot exclusion." Nexor Corp. `http://www.nexor.co.uk/mak/doc/robots/norobots.html`.

(Koster 1994b) Koster, Martijn. "Guide for robot writers." Nexor Corp. `http://www.nexor.co.uk/mak/doc/robots/guidelines.html`.

(Koster 1994c) Koster, Martijn. "List of robots." Nexor Corp. `http://www.nexor.co.uk/mak/doc/robots/active.html`.

(Koster 1994d) Koster, Martijn. "World Wide Web Robots, Wanderers, and Spiders." `http://www.nexor.co.uk/mak/doc/robots/robots.html`.

IPR2025-00714
DATABRICKS EX1008 Page 408

(Mauldin 1995) Mauldin, Michael L. "Measuring the Web with Lycos." Third International World Wide Web Conference, April 11, 1995. `http://lycos.cs.cmu.edu/lycos-websize.html`.

(McBryan 1994) McBryan, Oliver. "GENVL and WWWW: Tools for Taming the Web." Proceedings of the First International World Wide Web Conference. R. Cailliau, O. Nierstrasz, M. Ruggier (eds.), Geneva, 1994.

(ML 1994) Mauldin, Michael L. and John R. R. Leavitt. "Web Agent-Related Research at the Center for Machine Translation." SIGNIDR, McLean, Virginia, August 4, 1994. `http://fuzine.mt.cs.cmu.edu/mlm/signidr94.html`.

(Pinkerton 1994) Pinkerton, Brian. "Finding What People Want: Experiences with the WebCrawler." The Second International World Wide Web Conference '94: *Mosaic and the Web*. `http://webcrawler.com/WebCrawler/WWW94.html`.

(SM 1983) Salton, G. and M.J. McGill. "Introduction to Modern Information Retrieval." McGraw-Hill, New York, 1983.

(Salton 1989) Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

Chapter 5

(Asimov 1942) Asimov, I. "Runaround." *Astounding Science Fiction*. 1942.

(Eichmann 1994) Eichmann, D. "Ethical web agents." In *Proceedings of the Second International Conference on the World Wide Web*. Oct, 1994. Also available at `http://rbse.jsc.nasa.gov/eichmann/www-f94/ethics/ethics.html`.

(Koster 1994a) Koster, M. "A standard for robot exclusion." Nexor Corp. `http://web.nexor.co.uk/mak/doc/robots/norobots.html`.

(Koster 1994b) Koster, M. "Guide for robot writers." Nexor Corp. `http://web.nexor.co.uk/mak/doc/robots/guidelines.html`.

(Koster 1994c) Koster, M. "List of robots." Nexor Corp. `http://web.nexor.co.uk/mak/doc/robots/active.html`.

(EW 1994) Weld, D. and O. Etzioni. "The first law of robotics (a call to arms)." In *Proceedings of the Twelfth National Conference on AI*. Seattle, Wash. 1994. Also available at `ftp://cs.washington.edu/pub/ai/first-law-aaai-94.ps`.

Chapter 6

(Alvestrand 1995) Alvestrand, H. T.. "Tags for the identification of languages." RFC 1766, UNINETT, March 1995.

(AMLJTA 1993) Anklesaria, F. , M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti. "The Internet Gopher Protocol: A distributed document search and retrieval protocol." RFC 1436, University of Minnesota, March 1993.

(BL 1994) Berners-Lee. T. "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World Wide Web." RFC 1630, CERN, June 1994.

IPR2025-00714
DATABRICKS EX1008 Page 409

(BLC 1995) Berners-Lee, T. and D. Connolly. "HyperText Markup Language Specification—2.0." *Work in Progress* (draft-ietf-html-spec-01.txt). CERN, HaL Computer Systems, February 1995.

(BLMM 1994) Berners-Lee, T., L. Masinter, and M. McCahill. "Uniform Resource Locators (URL)." RFC 1738, CERN, Xerox PARC, University of Minnesota, October 1994.

(BF 1993) Borenstein, N. and N. Freed. "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies." RFC 1521, Bellcore, Innosoft, September 1993.

(Braden 1989) Braden, R. "Requirements for Internet hosts—application and support." STD 3, RFC 1123, IETF, October 1989.

(Crocker 1982) Crocker, D. H. "Standard for the Format of ARPA Internet Text Messages." STD 11, RFC 822, UDEL, August 1982.

(DKMSSWSG 1990) Davis, F., B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification." (v1.5), Thinking Machines Corporation, April 1990.

(KL 1986) Kantor, B. and P. Lapsley. "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News." RFC 977, UC San Diego, UC Berkeley, February 1986.

(Postel 1982) Postel, J. "Simple Mail Transfer Protocol." STD 10, RFC 821, USC/ISI, August 1982.

(Postel 1994) Postel, J. "Media Type Registration Procedure." RFC 1590, USC/ISI, March 1994.

(PR 1985) Postel, J. and J. K. Reynolds. "File Transfer Protocol (FTP)." STD 9, RFC 959, USC/ISI, October 1985.

(RP 1994) Reynolds, J. and J. Postel. "Assigned Numbers." STD 2, RFC 1700, USC/ISI, October 1994.

(ANSI 1986) US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

(ISO 1990) ISO-8859. International Standard—Information Processing—bit Single-Byte Coded Graphic Character Sets—Part 1: Latin Alphabet No. 1, ISO 8859-1:1987. Part 2: Latin alphabet No. 2, ISO 8859-2, 1987. Part 3: Latin alphabet No. 3, ISO 8859-3, 1988. Part 4: Latin alphabet No. 4, ISO 8859-4, 1988. Part 5: Latin/Cyrillic alphabet, ISO 8859-5, 1988. Part 6: Latin/Arabic alphabet, ISO 8859-6, 1987. Part 7: Latin/Greek alphabet, ISO 8859-7, 1987. Part 8: Latin/Hebrew alphabet, ISO 8859-8, 1988. Part 9: Latin alphabet No. 5, ISO 8859-9, 1990.

Chapter 7

(Fielding 1994) Fielding, Roy. "Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web." In *Proceedings of the First International Conference on the World Wide Web* (WWW 1994), Geneva, May 1994. Also available at `http://www.ics.uci.edu/WebSoft/MOMspider/WWW94/paper.html`.

(Tilton 1993) Tilton, James. "What is an Infostructure?" Available at `http://www.willamette.edu/~jtilton/info-p.html`.

IPR2025-00714
DATABRICKS EX1008 Page 410

Chapter 8

The Web Transaction Security Working Group (under the auspices of the Internet Engineering Task Force) is tasked with developing requirements and specifications for the provision of security services to HTTP. The Web Transaction Security Working Group has a mailing list at `mailto://www-security@nsmx.rutgers.edu`, as well as a home page at `http://www-ns.rutgers.edu/www-security/wts-wg.html`.

Rutgers University Network Service also maintains a home page on World Wide Web Security at `http://www-ns.rutgers.edu/www-security/index.html`.

The World Wide Web Consortium also maintains a home page listing the W3C Security Resources, and can be found at `http://www.w3.org/hypertext/WWW/Security/Overview.html`.

As an alternative, Phillip Hallam-Baker of CERN has proposed Shen as a security scheme for the World Wide Web, details of which are available at `http://www.w3.org/hypertext/WWW/Shen/ref/shen.html`.

(Bamford 1982) Bamford, James. *The Puzzle Palace: A Report on America's Most Secret Agency.* Houghton Mifflin, Boston. 1982.

(Brassard 1988) Brassard, G. "Modern Cryptography." Volume 325 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1988.

(CCITT 1988) CCITT (Consultative Committee on International Telegraphy and Telephony). Recommendation X.509: "The Directory—Authentication Framework." 1988.

(ElGamal 1985) ElGamal, T. "A public-key cryptosystem and a signature scheme based on discrete logarithms." IEEE Transactions on Information Theory. IT-31, pp.469–472, 1985.

(FIPS 1988) National Institute of Standards and Technology (NIST). FIPS Publication 46-1: Data Encryption Standard. January 22, 1988. Originally issued by National Bureau of Standards.

(Garfinkel 1995) Garfinkel, Simson. *PGP: Pretty Good Privacy.* O'Reilly and Associates, 1995.

(Hickman 1994) Hickman, Kipp E.B. Netscape Communications Corp. The SSL Prorocol. `http://home.mcom.com/info/SSL.html`.

(Hodges 1983) Hodges, Andrew. *Alan Turing: The Enigma.* Simon and Schuster, New York, 1983.

(Kahn 1967) Kahn, David. *The Codebreakers.* Macmillan, New York, 1967.

(Kahn 1983) Kahn, David. *Kahn on Codes: Secrets of the New Cryptology.* Macmillan, New York, 1983.

(Kahn 1991) Kahn, David. *Seizing the Enigma: The Race to Break the German U-Boat Codes, 1939–1943.* Houghton Mifflin, Boston. 1991.

(Kent 1993) Kent, S. RFC 1422: Privacy Enhancement for Internet Electronic Mail, "Part II: Certificate-Based Key Management." Internet Activities Board, February, 1993.

(Koblitz 1987) Koblitz, N. "Elliptic curve cryptosystems." *Mathematics of Computation.* 48:203–209, 1987.

IPR2025-00714
DATABRICKS EX1008 Page 411

(Linn 1993) Linn, J. RFC 1421: Privacy Enhancement for Internet Electronic Mail, "Part I: Message Encryption and Authentication Procedure." Internet Activities Board, February, 1993.

(LM 1991) Lai, XueJia and James Massey. "A proposal for a new block encryption standard." In *Advances in Crytology*—EuroCrypt '90 Proceedings. Springer-Verlag, 1991.

(Miller 1986) Miller, V.S. "Use of elliptic curves in cryptography." In *Advances in Cryptology*—Cryto '85, pp.417–426, Springer-Verlag, New York, 1986.

(NIST 1992) National Institute of Standards and Technology (NIST). "The Digital Signature Standard, proposal and discussion." *Commun. ACM.* 35(7), pp.36–54, July, 1992.

(Rivest 1990) Rivest, R.L. Cryptography. In J. van Leeuwen, ed. *Handbook of Theoretical Computer Science.* MIT Press/Elsevier, Amsterdam, 1990.

(RS 1994) Rescorla, Eric and Allan Schiffman. Enterprise Integration Technologies. "The Secure HyperText Transfer Protocol." 1994. `http://www.commerce.net/information/standards/drafts/shttp.txt`. Also available via e-mail at `mailto://shttp-info@commerce.net` or anonymous ftp at `ftp://ftp.commerce.net/pub/standards/drafts/shttp.txt`.

(RSA 1993) RSA Data Security. "Answers to Frequently Asked Questions About Today's Cryptography." Revision 2.0. October, 1993. Also available at `http://www.rsa.com/rsalabs/faq/faq_toc.html`.

(Schneier 1994) Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, New York. 1994.

(Schnorr 1989) Schnorr, C.P. "Efficient identification and signatures for smart cards." In *Advances in Cryptology*—Crypto '89, pp.239-251, Springer-Verlag, New York, 1990.

(SNS 1988) Steiner, J.G., B.C. Neuman, and J.I. Schiller. "Kerberos: an authentication service for open network systems." In Unix Conference Proceedings, pp.191–202, Dallas, Texas, February, 1988.

(US 1991) Comptroller General of the United States. "Matter of National Institute of Standards and Technology—Use of Electronic Data Interchange Technology to Create Valid Obligations." File B-245714, December 13, 1991.

Chapter 9

(Bauer 1994) Bauer, Paul W. "A Beginner's Guide to the U.S. Payments System," Economic Commentary, July 1, 1994.

(BC 1989) Brassard, G. and C. Crepeau. "Sorting out zero knowledge." In *Advances in Cryptology*—Eurocrypt '89 Proceedings. Springer-Verlag 1990.

(BR 1994) Borenstein, N.S. and M.T. Rose, "The application/green-commerce MIME Content-type." First Virtual Holdings Incorporated, October, 1994.

(Brands 1993) Brands, Stefan. "An efficient off-line electronic cash system based on the representation problem." Technical Report CSR9323, Computer Science Department, CWI, March 1993.

IPR2025-00714
DATABRICKS EX1008 Page 412

(Brands 1994) Brands, Stefan. "Untraceable off-line cash in wallet with observers." In *Advances in Cryptology*—CRYPTO '90 Proceedings. Springer-Verlag, 1991.

(Chaum 1982) Chaum, David. "Blind signatures for untraceable payments." In *Advances in Cryptography*—CRYPTO '82 Proceedings. Plenum Press, 1983.

(Chaum 1985) Chaum, David. "Security without identification: Transaction systems to make big brother obsolete." *Commun. ACM.* vol.28, no.10, pp.1030–1044, Oct 1985.

(Chaum 1992) Chaum, David. "Achieving electronic privacy." *Scientific American.* pp.96–101, August 1992.

(Econ 1994) "So much for the cashless society." *The Economist.* vol.333, issue 7891, pp.21–23, Nov 26, 1994.

(GS 1979) Garbade, Kenneth G. and William L. Silber. "The Payments System and Domestic Exchange Rates: Technological versus Institutional Change." *Journal of Monetary Economic.* vol.5, no.1 (Jan 1979), pp.1–22.

(RB 1994) Rose, M.T. and N.S. Borenstein, "The Simple Green Commerce Protocol (SGCP)." First Virtual Holdings Incorporated, October, 1994.

(Schneier 1994) Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, New York. 1994.

(White 1993) White, Peter T. "The Power of Money." *National Geographic.* vol.183, no.1 (Jan 1993), pp.80–108.

Chapter 10

(Brunner 1975) Brunner, John. *The Shockwave Rider.* Ballatine, New York. 1975.

(Cohen 1987) Cohen, Fred. "Computer Viruses—Theory and Experiments." *Computers and Security,* vol.6, no.1, pp.22–35, 1987.

(Cohen 1988) Cohen, Fred. "On the Implications of Computer Viruses and Methods of Defense." *Computers and Security,* vol.7, no.2, pp.167–184, 1988.

(Cohen 1989) Cohen, Fred. "Models of Practical Defenses Against Computer Viruses." *Computers and Security,* vol.8, no.2, pp.149–160, 1989.

(Cohen 1990) Cohen, Fred B. *A Short Course on Computer Viruses.* ASP Press, 1990.

(Ferbrache 1992) Ferbrache, David. *A Pathology of Computer Viruses.* Springer-Verlag, 1992.

(Gerrold 1972) Gerrold, David. *When Harley was One.* Doubleday, New York. 1972.

(Hoffman 1990) Hoffman, Lance J. (Ed.) *Rogue Programs: Viruses, Worms and Trojan Horses.* Van Nostrand Reinhold, 1990.

(Lucas 1994) Lucas, Kelly. "Viruses and Other Nasty Things." In *Cyberlife.* pp.423–449. Sams Publishing, 1994.

(Seeley 1989) Seeley, Donn. "A Tour of the Worm." In *Proceedings of the 1989 Winter Unix Conference.* USENIX Association, San Diego, California, Feb. 1989.

IPR2025-00714
DATABRICKS EX1008 Page 413

(SH 1982) Shoch, John F. and Jon A. Hupp. "The 'Worm' programs—Early experience with a distributed computation." *Commun. ACM*, vol.25, no.3, pp.172–180, March 1982.

(SHF 1990) Spafford, Eugene H., Kathleen A. Heaphy, and David J. Ferbrache. "A computer virus primer." In *Computers Under Attack: Intruders, Worms, and Viruses*. Editor: Peter Dennings. ACM Press/Addison-Wesley, 1990.

(Spafford 1989) Spafford, Eugene H. "The Internet worm: Crisis and aftermath." *Commun. ACM*, vol.32, no.6, pp.678–687, June 1989.

Chapter 11

The MUD Archive by Lauren P. Burka contains lots of MUD-related resources and can be found at `http://www.ccs.neu.edu/home/lpb/muddex.html`.

A current list of MUDs can be obtained via e-mail at `mailto://awozniak@geordi.calpoly.edu` with SUBSCRIBE as the subject. The MUD list also is available at `ftp://b63062.student.cwru.edu:/pub/mudlist/` or `http://b63062.student.cwru.edu/~mudlist`.

Further information about the Loebner Prize Competition may be obtained from Cambridge Center for Behavioral Studies, 675 Massachusetts Avenue, Cambridge, Mass., 02139. The Loebner Prize Home Page can be found at `http://www.csusm.edu/loebner_contest.html`.

The Colin Source Code, written by Michael Mauldin at CMU, can be found at `ftp://nl.cs.cmu.edu/usr/mlm/ftp/robot.tar.Z`.

Julia's log of the 1993 Loebner Prize Competition can be found at `http://fuzine.mt.cs.cmu.edu/mlm/contest93.html`.

Julia's log of the 1994 Loebner Prize Competition can be found at `http://fuzine.mt.cs.cmu.edu/mlm/contest94.html`.

Julia has a home page at `http://fuzine.mt.cs.cmu.edu/mlm/julia.html`.

To carry out a conversation with Julia, try `telnet://julia@fuzine.mt.cs.cmu.edu`.

Information about the CHAT Natural Language System can be found at `ftp://debra.dgbt.doc.ca/pub/chat/chat.html`. CHAT itself can be reached at `telnet://debra.dgbt.doc.ca:3000`.

(Bartle 1990) Bartle, Richard. "Interactive Multi-User Computer Games." MUSE Ltd. Research Report, December 1990. Also available at `ftp://parcftp.xerox.com:pub/MOO/papers/mudreport.txt`.

(Colby 1975) Colby, Kenneth M. *Artificial Paranoia: A Computer Simulation of Paranoid Process*. Pergamon Press., New York, 1975.

(Curtis 1992) Curtis, Pavel. "Mudding: Social Phenomena in Text-Based Virtual Realities." From 1992 Conference on Directions and Implications of Advanced Computing. `ftp://parcftp.xerox.com:pub/MOO/papers/DIAC92.txt`.

(CN 1993) Curtis, Pavel and David A. Nicholas. "MUDs Grow Up: Social Virtual Reality in the Real World." From The Third International Conference on Cyberspace. May, 1993. `ftp://parcftp.xerox.com:pub/MOO/papers/MUDsGrowUp.txt`.

IPR2025-00714
DATABRICKS EX1008 Page 414

(Epstein 1992) Epstein, R. "The Quest for the Thinking Computer." *AAAI Magazine*, Vol. 13, No. 2, Summer 1992, pp. 80–95.

(Epstein 1993) Epstein, R. "1993 Loebner Prize Competition in Artificial Intelligence: Official Transcripts and Results." Technical Report, Cambridge Center for Behavioral Studies, December 1993.

(Foner 1993) Foner, Leonard. "What's an Agent, Anyway? A Sociological Case Study." Agents Memo 93–01. Agents Group, MIT Media Lab. `http://foner.www.media.mit.edu/people/foner/Julia/`. `ftp://media.mit.edu/pub/Foner/Papers/Julia/Agents—Julia.ps`.

(Gibson 1984) Gibson, William. *Neuromancer.* Ace Science Fiction Books, New York. 1984.

(Mauldin 1994) Mauldin, Michael L. "Chatterbots, TinyMUDs, and the Turing Test: Entering The Loebner Prize Competition." AAAI-94. `http://fuzine.mt.cs.cmu.edu/mlm/aaai94.html`.

(Raymond 1991) Raymond, Eric S., editor, *The New Hacker's Dictionary.* MIT Press, 1991.

(Rheingold 1991) Rheingold, Howard. *Virtual Reality.* Summit Books, New York, 1991.

(Shieber 1992) Shieber, S. "Lessons from a Restricted Turing Test," Technical Report TR-19-92, Harvard University, Sept. 1992, Revision 4.

(Turing 1950) Turing, Alan M. "Computing Machinery and Intelligence." *Mind.* Vol. 54, No. 236, October 1950, pp. 433–460.

(Weizenbaum 1966) Weizenbaum, Joseph. "Eliza: a computer program for the study of natural language communication between man and machine." *Commun. ACM.* 9, 1, pp.36–44, 1966.

(Weizenbaum 1976) Weizenbaum, Joseph. *Computer Power and Human Reason.* W.H. Freeman and Co., New York, 1976.

(Wired 1993) Spence, Kristin and Louis Rossetto. *Wired Magazine*, San Francisco, Vol. 1.6, Dec. 1993. `http://www.wired.com/Etext/1.6/departments/net.surf.html`.

IPR2025-00714
DATABRICKS EX1008 Page 415

index

Symbols

/ (slash), URIs, 68
? (question mark), Web address queries, 68

A

AberMUDs, 256
Accept header (HTTP request messages), 133-134
Accept-Charset header (HTTP request messages), 134
Accept-Encoding header (HTTP request messages), 134
Accept-Language header (HTTP request messages), 134-135
access security, 186
ACDS (Automated Catalog Design Service), 27
ACL (Agent Communication Language), 15-16
addresses (WWW), 67-69
Advanced Research Projects Agency (ARPA), 38
Agent Communication Language (ACL), 15-16
agents, 5-6
  application agents, 13
  architectural agents, 24-25
  as personal assistants, 6-7
  automated design agents, 27
  BargainFinder, 77
  BookFinder, 347-349
  Calendar Apprentice, 19-21
  catalog agents, 27
  CDFinder, 351-353
  COACH, 17-19
  commercial development, 34
  common-sense agents, 21-23
  communicative agents, 15-16
  conference agents, 12-13
  constraint agents, 27
  creative agents, 24-26
  Cyc project, 22-23
  design agents, 25-26
  emotional characteristics, 27-30
  Envoy agents, 8
  facilitators, 16
  floor agents, 13
  group agents, 13
  information commerce, 221
  integrated agents, 13-15
  intelligence levels, 35
  jazz agents, 25
  KidSim, 30-31
  learning agents, 19-21
  mathematical agents, 25
  MUDs, 250
  New Wave Agent, 8-9
  Oasis agents, 31
  physical agents, 23-24
  programming, 16, 30-32
  server authentication, 135, 151, 290
  societal issues, 32-33
  softbots, 10-11
  software agents, 15
  surrogate bots, 9-10
  system agents, 27
  teaching agents, 17-19
  Telescript programming language, 5
  Turing test, 250-251
  user agents, 13
  visitorbots, 9-10
  WebCrawler, 88
  WebShopper, 78
    downloading, 348
    program listing, 337-345
  WWW, 76-78
Alameda virus, 239
Allow header, 142
ANS (Advanced Network Services), 42
application agents, 13
Arachnophilia spider, 381
architectural agents, 24-25
architecture (WebCrawler), 86
archiving robot Web search results, 120
Aretha robot, 385

IPR2025-00714
DATABRICKS EX1008 Page 417

armored viruses, 246
ARPA (Advanced Research Projects Agency), 38
ARPAnet, 38-39
artificial intelligence, 28
  Eliza chatterbot, 251-253
  Julia chatterbot, 273-279
  Loebner Prize Competition, 275
  Parry chatterbot, 253-255
Asimov, Isaac, 122
ASpider (Associative Spider), 379
augmented BNF (HTTP), 284-285
AUP (acceptable-use policy), Internet, 42
authenticating e-mail transactions, 188-189
Authorization header (HTTP request messages), 135
Automated Catalog-Design Service (ACDS), 27
automated design agents, 27
avoidance package (WebWalker program), 174-175

B

backend search engines, 98
Backus-Naur Form (BNF), 284-285
BargainFinder agent, 77
$BaseURL parameter (WebWalker), 173
$BetweenTime parameter (WebWalker), 173
BITNET (Because It's Time Network), 39
blind signatures (digital cash), 210-211
BNF (Backus-Naur Form), 284-285
Book Stacks Unlimited (online bookstore), 348
BookFinder agent, 347-349
bookstores (online), 347-349
boot-sector infector viruses, 236
bots, *see* robots
BoundURL, URL prefix (WebWalker directive), 160
Brain virus, 242-243
Broker (Harvest), 97-98
browsers (WWW), 66
business-related agents
  Calendar Apprentice agent, 19-21
  communicative agents, 15-17
  conference agents, 12-13
  Envoy Framework, 8
  integrated agents, 13-15
  New Wave desktop agent, 8-9
  personal assistants, 6-7
  surrogate bots, 9-10
  visitor scheduling (visitorbots), 9-10

C

caching Web documents, 67
Calendar Apprentice (CAP) agent, 19-21
catalog agents, 27
CD Land (online music store), 352
CD World (online music store), 352
CDFinder agent, 351-353
CDNow (online music store), 352
CERN servers, 66
CGI (Common Gateway Interface), 74-75, 167-169
ChangeWindow number (WebWalker directive), 160
character sets
  HTTP entities (preferences), 148-149
  HTTP parameters, 148-149
  HTTP request messages (preferences), 134
CHAT chatterbot, 278
chatterbots (MUDs), 4, 250
  CHAT, 278
  Eliza, 251-253
  Julia, 273-279
  Parry, 253-255
  tricks, 278-279
Checkbot, 382
Christmas Tree worm, 232
ChURL robot, 381
client programs
  MUDs, 256
  WWW, 66
  see also browsers
client/server computing, 126-127
cluster viruses, 237
COACH (Cognitive Adaptive Computer Help) agent, 17-19
Colin robot (TinyMUDs), 269-270
  information services, 270-271
  mapping services, 271-273
  miscellaneous services, 273
commerce agents, 4, 76
  BargainFinder, 77
  WebShopper, 78
commercial transactions, 207
  digital cash, 207-214
  information, 220-225
  via Internet, 214-220
Common Gateway Interface (CGI), 66, 74-75

IPR2025-00714
DATABRICKS EX1008 Page 418

common-sense agents, 21-23
communicative agents, 15-16
companion viruses, 236
compiler systems (Oasis), 32
CompuBooks (online bookstore), 348
Computer Literacy Bookshops (online bookstore), 348
computer viruses, *see* viruses
conference agents, 12-13
configurations (WebWalker program), 172-174
constraint agents, 27
content negotiation (HTTP), 150-151
content parameters (HTTP), 148-150, 290
Content-Encoding header, 142
Content-Language header, 142-143
Content-Length header, 143
Content-Transfer-Encoding header, 143
Content-Type header, 143
controlling robots, 118
creative agents, 24-26
credit card services (CyberCash), 217
CREN (Corporation for Research and Education Networking), 41
cryptography, 189-191
  key management, 192
  public-key, 191-194
  secret-key, 191-192
cryptosystems, 187
  public-key, 193
  RSA public-key, 192-193
  secret-key, 193
CS-HKUST WWW Index Server, 384
CSNET (Computer Science Research Network), 39
CTE (Content-Transfer-Encoding) header, 143
Cyberbucks, 212
CyberCash, 215
  credit card services, 217
  money payments service, 219-220
  payment options, 216-217
cybershops, 212
cyberspace, 249
Cyc project (agents), 22-23

D

Dark Avenger virus, 241-244
DARPA (Defense Advanced Research Project Agency), 38
Darth Vader virus, 241
Data Encryption Standard (DES), 191
data integrity (e-mail transactions), 189
data types (HTTP entities), 146-147
databases (WebCrawler), 84, 88
Date header (HTTP messages), 129
date/time stamps (HTTP applications), 147
dead links, 107, 154
Defense Advanced Research Project Agency (DARPA), 38
DELETE method (HTTP request messages), 133
Derived-From header, 143-144
DES (Data Encryption Standard), 191
design agents, 25-26
desktop agents
  Envoy Framework, 8
  New Wave Agent, 8-9
detecting viruses, 241-242
Df (document frequency), 92
digital cash, 206-208
  anonymity, 210-211
  blind signatures, 210-211
  CyberCash, 215-220
  Ecash, 211-214
  security, 210-211
  unconditional untraceability, 210
digital cashier's checks, 208-209
Digital Signature Algorithm (DSA), 197
digital signatures, 188-189, 194-197
DikuMUD, 256
Disallow records (robot exclusion files), 110
document frequency (Df), 92
documents (HTML), 72
  indexing, 100-103
    with Harvest Gatherer/Broker, 96-99
    with Lycos, 92
    with WebAnts, 100
    with WebCrawler, 85
  maintenance with robots, 107
  parent, 112
  peculiarity weighting, 88
  storing with Harvest Object Cache, 98
  Web page referrers, 112
domains (Internet), 41, 44
Doran's MUDlist, 356-374
downloading WebShopper, 348
DSA (Digital Signature Algorithm), 197

IPR2025-00714
DATABRICKS EX1008 Page 419

E

e-mail
  robot operator addresses, 159
  transactions
    authentication, 188-189
    data integrity, 189
    digital signatures, 194-197
    key certification, 197-198
    security, 187
Ecash, 211-214
Eichmann, David, 123
EIT Link Verifier Robot, 380
electronic banking, 214-220
electronic commerce, 76-78, 225-226
Eliza (chatterbot), 251-253, 278
Emacs W3 Search Engine robot, 380
EMusic (online music store), 353
encoding mechanisms (HTTP), 149
encryption, 187-191
  key certification, 197-198
  public-key cryptography, 191-194
  secret-key cryptography, 191-192
  viruses, 245-246
entities (HTTP), 141, 288-289
  Allow header, 142
  body, 146-147, 289
  character set preferences, 148-149
  Content-Encoding header, 142
  Content-Language header, 142-143
  Content-Length header, 143
  Content-Transfer-Encoding header, 143
  Content-Type header, 143
  data types, 146-147
  Derived-From header, 143-144
  encoding mechanisms, 149
  Expires header, 144
  headers, 288-289
  Last-Modified header, 144
  Link header, 144-145
  Location header, 145
  transfer encoding, 149-150
  URI header, 145
  Version header, 145-146
Envoy agents, 8
errors, reporting during robot Web searches, 115
ethics (robots), 112, 122-123
Etzioni, Oren, 122
Exclude, URL prefix (WebWalker directive), 160
excluding
  robots from Web sites, 108-110, 122
  URLs from WebWalker traversals, 156
exclusion files (robots), 108-110, 156-157
Expires header, 144
ExpireWindow number (WebWalker directive), 160

F

facilitators (agents), 16
fast infector viruses, 241
FidoNet, 40
file infector viruses, 236
fill out forms (HTML), 73
fingering robot operators, 121
firewalls, 67
First Virtual payment
  Green Commerce model, 223
  information commerce, 222-226
  security, 223-225
  transactions, 223-224
Fish search engine, 87, 377
floor agents, 13
fork( ), worm defense mechanism, 233
formats
  HTTP negotiations, 70-71
  HTTP request messages, 130
  public-key certification, 198
  robot exclusion files, 109
  WebWalker task files, 161
forms (HTML), 73
Forwarded header (HTTP messages), 129
Four Laws of Web Robotics, 110-115
Friday the 13th virus, 243
Frodo virus, 241
From header (HTTP request messages), 135
FTP
  objects, storing with Harvest Object Cache,'98
  sites
    EIT Link Verifier Robot, 380
    HTMLgobble robot, 378
    mirroring, 107
    MOMspider, 181, 377-378
    Python Robot, 377
    searching with Harvest Gatherer, 97
    searching with Lycos, 91
    WebCopy robot, 385
    Weblayers robot, 385

IPR2025-00714
DATABRICKS EX1008 Page 420

functions (WebWalker), 169-170
avoidance package, 175
history package, 176
summary package, 180-181
traversal package, 177-178
Future Fantasy Bookstore (online bookstore), 348

G

gateways, 74-75
Gatherer (Harvest), 96-97
Genghis robot, 23
GET method (HTTP request messages), 131
GetURL robot, 383
Gibson, William, 249
Glimpse backend search engine, 98
global directives (WebWalker), 159
Global Electronic Music Marketplace (online music store), 352
Gopher, 69
objects, storing with Harvest Object Cache, 98
sites
searching with Harvest Gatherer, 97
searching with Lycos, 91
versus WWW, 66
Green Commerce model (information commerce), 223
group agents, 13

H

hard disks (architecture), 237
Harvest, 382
architecture, 95-99
backend search engines, 98
Broker, 97-98
Gatherer, 96-97
Home Pages Broker, 94
Object Cache, 98
registering, 97
Replicator, 99
Server Registry (HSR), 97
HEAD method (HTTP request messages), 131
headers (HTTP)
entities, 141-146, 288-289
messages, 128, 286
request messages, 133-137, 287-288
response messages, 140-141, 288
Hewlett-Packard New Wave Agent, 8-9
history package (WebWalker program), 175-176
home pages, *see* WWW, sites
Home Pages Broker (Harvest), 94
HSR (Harvest Server Registry), 97
HTML (HyperText Markup Language), 67
forms, 73
levels of SGML conformance, 71
tags, 72
HTML Analyzer, 377
HTMLgobble robot, 378
HTTP (HyperText Transfer Protocol), 67-71, 125-126
augmented BNF, 284-285
character set parameters, 148-149
client/server computing, 126-127
content negotiation, 150-151
content parameters, 148-150, 290
date/time stamps, 147
encoding mechanisms, 149
entities, 141, 288-289
Allow header, 142
body, 146-147, 289
character sets, 148-149
Content-Encoding header, 142
Content-Language header, 142-143
Content-Length header, 143
Content-Transfer-Encoding header, 143
Content-Type header, 143
data types, 146-147
Derived-From header, 143-144
Expires header, 144
headers, 288-289
Last-Modified header, 144
length of body, 147
Link header, 144-145
Location header, 145
URI header, 145
Version header, 145-146
format negotiations, 70-71
language tags, 150
Media Type parameters, 148
messages, 128
Date header, 129
Forwarded header, 129
headers, 128, 286
Message-ID header, 129-130
MIME-Version header, 130
posting, 131
parsing constructs, 285-286
protocol parameters, 147-148, 289-290

IPR2025-00714
DATABRICKS EX1008 Page 421

request messages, 130-136, 286-288
response messages, 136-141, 287-288
transactions, 70
transfer encodings, 149-150
URI parameter, 147
Version parameter, 147
hypertext, 71
dead links, 107, 154
gateways, 75
HyperText Markup Language, *see* HTML
HyperText Transfer Protocol, *see* HTTP

I

i? music/media (online music store), 352
IAB (Internet Activities Board), 56
IDEA (International Data Encryption Algorithm), 191
identifying, 111-112, 116-117, 234
IDf (inverse document frequency), 92
IETF (Internet Engineering Task Force), 55-56
If-Modified-Since header (HTTP request messages), 135
imagemaps, 73-74
IMPs (Interface Message Processors), 38
indexing Web documents, 83, 100-103
with Harvest Gatherer/Broker, 96-99
with Lycos, 92-93
with WebAnts, 100
with WebCrawler, 85-89
information commerce
agents, 221
economics, 220-221
First Virtual payment system, 222-226
security, 221-222
InfoSeek Robot, 383
installation (WebWalker), 158-159
instructions rescheduling virus, 246
integrated agents, 13-15
Interface Message Processors (IMPs), 38
interfaces, 66, 74-75, 167-169
International Data Encryption Algorithm (IDEA), 191
Internet
AUP (acceptable-use policy), 42
commercial transactions, 214-220, 225-226
commercialization, 42
domains, 41, 44
early development, 38-39
government regulation, 57-60
growth, 42-44
information commerce, 220-225
international use, 45-54
monetary transactions, 207
privacy issues, 187
searching with softbots, 11
security, 186-189
softbots, 10-11
users, 45-54
Internet Activities Board (IAB), 56
Internet Engineering Task Force (IETF), 55-56
Internet Society (ISOC), 55-56
Internet worm, 232
Intertain.com (online bookstore), 348
inverse document frequency (IDf), 92
IP (Internet Protocol), 39
ISOC (Internet Society), 55-56

J–K

jazz agents, 25
Julia chatterbot, 273-279
JumpStation Robot, 376

Katipo robot, 382-383
key certification (encryption), 197-198
key management (cryptography), 192
KidSim, 30-31
KIF (Knowledge Interchange Format), 15
KQML (Knowledge Query and Manipulation Language), 15

L

Lahore virus, 242
language tags (HTTP), 150
Last-Modified header, 144
learning agents, 19-21
Lehigh virus, 240, 243
$LibWWW parameter (WebWalker), 172
limiting WebWalker traversal depth, 160
Link header, 144-145
LINK method (HTTP request messages), 133
links (gateways), 75
Lisp programming language (COACH agent), 18
List of Robots Web site, 106
listings (programs)
WebShopper, 337-345
WebWalker, 294-335
Listservs, 39
$LocalNetwork (WebWalker), 172

IPR2025-00714
DATABRICKS EX1008 Page 422

Location header, 145
Loebner Prize Competition, 275
log files
  robots, 118
  WebWalker traversals, 157, 175-176
  WWW sites, 154
loops (robot Web searches), 114
LPMUD, 256
Lycos robot, 88-93, 379

M

M integrated agent, 13-15
Mac WWWWorm robot, 381
Macmillan Bookstore (online bookstore), 348
manual traversal (Web sites), 155
Martijn Koster's List of Active Robots, 375
master boot records (hard disks), 237
mathematical agents, 25
$MaxConsec parameter (WebWalker), 173
$MaxDepth parameter (WebWalker), 173
MaxDepth number (WebWalker directive), 160
Media Type parameters (HTTP), 148
Message-ID header (HTTP messages), 129-130
messages (HTTP), 128-131, 286
methods (HTTP request messages), 130-133, 287
Michelangelo virus, 244
MIME (Multi-purpose Internet Mail Extensions), 71, 126
MIME-Version header (HTTP messages), 130
mirroring, 99, 108
MOMspider (Multi-Owner Maintenance Spider), 156, 181, 377-378
money payments service (CyberCash), 219-220
money, *see* digital cash
MOOs, 267, 368-369
MPs (message processors), 38
MtE (Mutation Engine) virus, 246
MUCKs, 267, 369-370
MUDs (Multi-User Dungeons), 250, 356-374
  agents, 4, 250
  chatterbots, 250
    Eliza, 251-253
    Julia, 273-279
    Parry, 253-255
  client programs, 256
  components, 256
  Doran's MUDlist, 356-374
  early development, 255-256
  robots, 250, 256
  sample interaction, 256-266
  wizards, 256
  see also TinyMUDs
multi-encrypted viruses, 246
Multi-Owner Maintenance Spider (MOMspider), 156, 181, 377-378
Multi-purpose Internet Mail Extensions (MIME), 71
Multi-User Dungeons, *see* MUDs
multipartite viruses, 237
MUSHes (Multiple User Shared Hallucinations), 267, 371-372
Music Connection (online music store), 353
music stores (online), 351-353
Mutation Engine (MtE) virus, 246

N

Name infostructure_name (WebWalker directive), 160
NAPs (Network Access Points), 42
National Information Infrastructure (NII), 57-60
navigating WWW
  with Lycos, 92-93
  with WebCrawler, 85-89
NCP (Network Control Protocol), 38
NCSA servers, 66
Nebula backend search engine, 98
networks
  BITNET, 39
  CREN, 41
  CSNET, 39
  FidoNet, 40
  NSFnet, 41-42
  USENET, 40
  worms, 230-232
New Wave Agent, 8-9
newsgroups, 40, 117
NHSE Web Forager robot, 380
NII (National Information Infrastructure), 57-60
NIKOS robot, 383
NNTP (Network News Transfer Protocol), 40
NorthStar Robot, 376-377
Noteworthy Music (online music store), 353
NSFnet (National Science Foundation network), 41-42

IPR2025-00714
DATABRICKS EX1008 Page 423

O

Oasis (Object and Agent Specification and Implementation System), 31-32
Object Caches (Harvest), 98
online bookstores, 347-349
online music stores, 351-353
Open Text Corporation Robot, 383
operator identification (robots), 112
output (WebWalker), 162-167
Oz Project, 28-30

P

Pakistani virus, 242
parameters (HTTP)
  content, 148-150, 290
  protocol, 147-148, 289-290
parent documents (WWW), identifying, 112
Parry (chatterbot), 253-255, 279
parsing constructs (HTTP), 285-286
partition records (hard disks), 237
$PauseTime parameter (WebWalker), 173
peculiarity weighting (indexing Web documents), 88
PEM (Privacy-Enhanced Mail) format, 198
Peregrinator robot, 381-382
Perl programming language, 158
physical agents, 23-24
polymorphic virus, 245-246
ports (Web servers), 68
POST method (HTTP request messages), 131-132
posting HTTP messages, 131
Powell's Technical Books (online bookstore), 349
Pragma header (HTTP request messages), 135-136
precision (spiders), 101
Privacy-Enhanced Mail (PEM) format, 198
processing tasks (WebWalker program), 174
program listings
  WebShopper, 337-345
  WebWalker, 294-335
programming agents, 16, 30-32
program gateways, 74-75
protocols
  HTTP, 67, 69-71, 125-130
  IP, 39
  NCP, 38
  NNTP, 40
  parameters (HTTP), 147-148, 289-290
  S-HTTP, 201-203
  SSL, 200-201
  TCP, 39
proxy servers, 67, 129
Public header (HTTP response messages), 141
public-key cryptography, 191-194
public-key cryptosystems, 193, 197-198
publishing robot Web search results, 120
PUT method (HTTP request messages), 132-133
Python Robot, 377

Q–R

queries
  rotating during robot Web searches, 114
  URLs, 68
query servers (WebCrawler), 88
question mark (?), Web address queries, 68

RBSE (Repository Based Software Engineering) spider, 83, 376
reason phrases (HTTP response messages), 137-140, 288
recall (spiders), 101
Referrer header (HTTP request messages), 136
registering Harvest components, 97
Reiters Books (online bookstore), 349
replicating indexed Web documents, 99
Replicator (Harvest), 99
ReplyTo email_address (WebWalker directive), 159
reports (WebWalker), 157
Request for Comments, *see* RFCs
request messages (HTTP), 130, 286-287
  formats, 130
  headers, 287-288
    Accept, 133-134
    Accept-Charset, 134
    Accept-Encoding, 134
    Accept-Language, 134-135
    Authorization, 135
    From, 135
    If-Modified-Since, 135
    Pragma, 135-136
    Referrer, 136
    User-Agent, 136

IPR2025-00714
DATABRICKS EX1008 Page 424

methods, 287
DELETE method, 133
GET method, 131
HEAD method, 131
LINK method, 133
POST method, 131-132
PUT method, 132-133
UNLINK method, 133
resources
conserving when using robots, 113-115
discovering with robots, 107
response messages (HTTP), 136-137, 287-288
headers, 288
Public, 141
Retry-After, 141
Server, 141
WWW-Authenticate, 141
reason phrases, 137-140, 288
status codes, 137-140, 288
restricting WebWalker traversals, 174-175
Retry-After header (HTTP response messages), 141
RFCs (Request for Comments), 56
robots, 4, 78
Aretha, 385
Checkbot, 382
ChURL, 381
Colin (TinyMUDs), 269-270
controlling, 118
CS-HKUST WWW Index Server, 384
EIT Link Verifier, 380
Emacs W3 Search Engine, 380
ethics, 112
etiquette, 113-120
excluding from Web sites, 108-110, 122
exclusion files, 108-110, 156-157
Fish Search, 377
Genghis, 23
GetURL, 383
Harvest, 93-99, 382
HTMLgobble, 378
identification, 111-112, 116-117
InfoSeek, 383
JumpStation, 376
Katipo, 382-383
Loebner Prize Competition, 275
log files, 118
Lycos, 89-93, 379
Mac WWWWorm, 381
mirroring Web sites, 108
MOMspider, 377-378
MUDs, 250, 256
NHSE Web Forager, 380
NIKOS, 383
NorthStar, 376-377
Open Text Corporation, 383
operator identification, 112
operator-to-Webmaster communication, 119
operators, 121, 159
Peregrinator, 381-382
Python, 377
resources, 107, 113-115
Scooter, 385
self identification, 111
SG-Scout, 379
Six Commandments for Robot Operators, 115-120
softbots, 10-11, 123
Spry Wizard, 384-385
surrogate bots, 9-10
Tcl W3, 384
testing, 117
TITAN, 384
TkWWW, 384
Turing test, 250-251
visitorbots, 9-10
Web ethics, 122-123
Web searches, 113-115, 120
Web site restrictions, 119-120
WebAnts, 99-100
WebCopy, 385
WebCrawler, 84-89
Webfoot, 379
Weblayers, 385
WebLinker, 380
Webmaster tips, 121-122
Webwalk, 382
WebWalker, 156-181
WebWatch, 385-386
WM32, 378
World Wide Web Wanderer, 377
World Wide Web Worm, 82, 378
see also spiders
robots.txt files (Web sites), 101
$RobotsURL parameter (WebWalker), 173
Roswell Internet Computer Bookstore (online bookstore), 349
rotating queries during robot Web searches, 114
RSA public-key cryptosystem, 192-193
running WebWalker, 156

IPR2025-00714
DATABRICKS EX1008 Page 425

S

S-HTTP (Secure HTTP), 201-203
scalability (spiders), 102-103
scheduling with visitorbots, 9
Scooter robot, 385
search engines
  Fish, 87
  Glimpse, 98
  Nebula, 98
  WebCrawler, 86-89
searching
  FTP sites
    with Harvest Gatherer, 97
    with Lycos, 91
  Gopher sites
    with Harvest Gatherer, 97
    with Lycos, 91
  Internet with softbots, 11
  Web sites (robot etiquette), 110-115
  WWW, 85-89, 91, 94, 100
secret-key cryptography, 191-193
sectors (hard disks), 237
Secure HTTP (S-HTTP), 201-203
Secure Socket Layer (SSL) protocol, 200-201
security
  access security, 186
  digital cash, 210-211
  e-mail transactions, 187
  Ecash, 213-214
  First Virtual payment service, 223-225
  information commerce, 221-222
  transaction security, 186
  WWW, 198-204
self identification (robots), 111
Server header (HTTP response messages), 141
servers, 66-67
  agent authentication, 135, 151, 290
  alternative ports, 68
  CERN, 66
  conserving resources when using robots, 113
  excluding robots from, 108-110
  NCSA, 66
  proxies, 67
  reducing overload, 101-102
  request messages, 130-136
  rotating robot queries, 114
service providers
  NAPs (Network Access Points), 42
setup (WebWalker traversals), 173-174
SG-Scout robot, 379
SGML (Standard Generalized Markup Language), 71
signatures (viruses), 241
Six Commandments for Robot Operators, 115-120
slash (/), URIs, 68
slow infector viruses, 241
softbots, 10-11, 123
Softpro Books (online bookstore), 349
software agents, 15
sparse infector viruses, 241
spiders
  Arachnophilia, 381
  ASpider, 379
  future developments, 103
  Lycos, 89-93
  MOMspider, 156, 181, 377-378
  precision, 101
  RBSE, 83, 376
  recall, 101
  scalability, 102-103
  Tarspider, 381
  Web maintenance, 107, 155-156
  WebCrawler, 84-89, 376
  WebWalker, 156-181, 294-335
  World Wide Web Worm, 82, 378
  see also robots
Spry Wizard Robot, 384-385
SSL (Secure Socket Layer) protocol, 200-201
Standard for Robot Exclusion, 108-110
Standard Generalized Markup Language (SGML), 71
startup (WebWalker), 161-162
statistical reports (WebWalker traversals), 178-181
status codes (HTTP response messages), 137-140, 288
stealth viruses, 244-245
Stoned virus, 238
storing, 98
subroutines (WebWalker program), 169-172
summary package (WebWalker program), 178-181
surrogate bots, 9-10
system agents, 27

T

tags (HTML), 72
Tarspider, 381

IPR2025-00714
DATABRICKS EX1008 Page 426

task directives (WebWalker), 160-161
task files (WebWalker), 159-161
$Taskfile parameter (WebWalker), 172
Tcl W3 Robot, 384
TCP (Transmission Control Protocol), 39
TCP/IP (Transmission Control Protocol/Internet Protocol), 41-42
teaching agents, 17-19
Telescript programming language (agents), 5
term frequency (Tf), 92
testing robots, 117
Tf (term frequency), 92
$Timeout parameter (WebWalker), 173
TinyMUDs, 250, 266-273
  see also MUDs
TITAN robot, 384
TkWWW Robot, 384
TopURL URL (WebWalker directive), 160
Tower Records (online music store), 353
tracks (hard disks), 237
transaction security, 186
transfer encodings (HTTP), 149-150
Transmission Control Protocol/Internet Protocol, *see* TCP/IP
traversal package (WebWalker program), 177-178
traversing Web sites with WebWalker, 156
Turing test, 250-251

U

unconditional untraceability (digital cash), 210
Uniform Resource Locators, *see* URLs
Universal Resource Identifiers, *see* URIs
UNLINK method (HTTP request messages), 133
updating Web sites, 154
URIs (Universal Resource Identifiers), 67-69, 145-147
URLs (Uniform Resource Locators), 67-69
  avoiding during robot Web searches, 113
  excluding from WebWalker traversals, 156
USENET (Users' Network), 40
user agents, 13
User-Agent header (HTTP request messages), 136
User-Agent records (robot exclusion files), 109

V

variables, 174-179
$Version parameter (WebWalker), 172
Version header (HTTP entities), 145-146
Version parameter (HTTP), 147
viruses, 4, 230, 241, 247
  Alameda, 239
  armored, 246
  boot-sector infector, 236
  Brain, 242-243
  cluster, 237
  companion, 236
  Dark Avenger, 241-244
  Darth Vader, 241
  detecting, 241-242
  encryption, 245-246
  fast infector, 241
  file infector, 236
  Friday the 13th, 243
  Frodo, 241
  history, 234-235
  instruction rescheduling, 246
  Lehigh, 240, 243
  life cycle, 238
  manifestations, 238
  Michelangelo, 244
  MtE, 246
  multi-encrypted, 246
  multipartite, 237
  polymorphic, 245-246
  stealth, 244-245
  Stoned, 238
visitorbots, 9-10

W

WAIS (Wide Area Information Server), 66, 69
Web maintenance spiders, 107
Web page referrers, 112
WebAnts, 99-100
WebCopy robot, 385
WebCrawler, 84-89, 376
Webfoot robot, 379
Weblayers robot, 385
WebLinker robot, 380
Webmasters (robot tips), 121-122
WebShopper agent, 78, 337-345, 348
Webspace, 154-156

IPR2025-00714
DATABRICKS EX1008 Page 427

WebWalker, 382
avoidance package, 174-175
$BaseURL parameter, 173
$BetweenTime parameter, 173
configurations, 172-174
CGI configurations, 167-169
global directives, 159
history package, 175-176
installation, 158-159
$LibWWW parameter, 172
limitations, 158
$LocalNetwork, 172
$MaxConsec parameter, 173
$MaxDepth parameter, 173
output, 162-167
$PauseTime parameter, 173
performance evaluation, 157
processing tasks, 174
program listing, 294-335
reports, 157
$RobotsURL parameter, 173
running, 156
source code (Web sites), 159
startup, 161-162
subroutine call graph, 170-172
subroutines, 169-170
summary package, 178-181
task directives, 160-161
task files, 159-161
$Taskfile parameter, 172
$Timeout parameter, 173
traversals, 177-178
date specifications, 160
excluding URLs, 156
limiting depth, 160
log files, 157, 175-176
restricting, 174-175
setup, 173-174
statistical reports, 178-181
URL boundaries, 160
URL exclusions, 160
URL specifications, 160
Webspace specifications, 160
variables, 174
$Version parameter, 172
WebWatch robot, 385-386
Weld, Daniel, 122
wizards (MUDs), 256
WM32 Robot, 378
World Wide Web, *see* WWW
World Wide Web Wanderer, 106, 377
World Wide Web Worm, 82, 378
worms, 4, 230, 247, 230-234
WWW (World Wide Web)
addresses, 67
architecture, 65-67
browsers, 66
client programs, 66
commerce agents, 76-78
commercialism, 78
documents
caching, 67
indexing, 100-103
indexing with Harvest Gatherer/Broker, 96-99
indexing with Lycos, 92
indexing with WebAnts, 100
indexing with WebCrawler, 85
peculiarity weighting, 88
storing with Harvest Object Cache, 98
Web page referrers, 112
early development, 62-65
formats, 67
functions, 65
growth, 62
indexing, 83
information dissemination, 62-65
maintenance with spiders, 155-156
monetary transactions, 207
navigating, 85-89, 92-93
privacy issues, 187
protocols, 67
proxy servers, 67
resource naming, 67
robots.txt files, 101
searches
archiving results, 120
avoiding URLs, 113
error reports, 115
file format compatibility, 113
loops, 114
publishing results, 120
query rotation, 114
robot etiquette, 110-115
server access times, 114
server resource conservation, 114
skipping query interfaces, 114
specifying, 113
with Harvest, 94
with Lycos, 91

IPR2025-00714
DATABRICKS EX1008 Page 428

with WebAnts, 100
with WebCrawler, 85-89
security, 198-204
servers, 66-67
alternative ports, 68
conserving resources when using robots, 113
reducing overload, 101, 102
rotating robot queries, 114
sites
Arachnophilia spider, 381
ASpider, 379
Book Stacks Unlimited, 348
BookFinder agent, 347
CD Land, 352
CD World, 352
CDFinder agent, 351
CDNow, 352
Checkbot, 382
ChURL robot, 381
CompuBooks, 348
Computer Literacy Bookshops, 348
CS-HKUST WWW Index Server, 384
cybershops, 212
Doran's MUDlist, 356-374
EIT Link Verifier Robot, 380
Emacs W3 Search Engine robot, 380
EMusic, 353
excluding robots from, 108-110, 122
Fish Search robot, 377
Future Fantasy Bookstore, 348
GetURL robot, 383
Global Electronic Music Marketplace, 352
Green Commerce model, 223
Harvest, 382
i? music/media, 352
InfoSeek Robot, 383
Intertain.com, 348
Julia chatterbot, 274
JumpStation Robot, 376
Katipo robot, 382-383
List of Robots, 106
Lycos robot, 379
Mac WWWWorm robot, 381
Macmillan Bookstore, 348
maintenance with robots, 107
manual traversal, 155
Martijn Koster's List of Active Robots, 375
mirroring, 108
MOMspider, 81, 377-378
Music Connection, 353
NIKOS robot, 383
NorthStar Robot, 376
Noteworthy Music, 353
Peregrinator robot, 381-382
Perl archives, 158
Powell's Technical Books, 349
Python Robot, 377
RBSE Spider, 376
Reiters Books, 349
robot restrictions, 119-120
Roswell Internet Computer Bookstore, 349
Scooter robot, 385
SG-Scout robot, 379
Softpro Books, 349
Spry Wizard Robot, 384-385
Standard for Robot Exclusion, 108
Tarspider, 381
Tcl W3 Robot, 384
TITAN robot, 384
TkWWW Robot, 384
Tower Records, 353
update log files, 154
WebAnts, 99
WebCrawler, 376
Webfoot Robot, 379
WebLinker robot, 380
WebShopper, 348
WebWalker source program, 159
WebWatch robot, 385-386
WM32 Robot, 378
World Wide Web Wanderer, 377
WORLD WIDE WEB WORM, 378
transactions (security), 203-204
traversing with WebWalker, 156-157
versus Gopher, 66
versus WAIS, 66
Webspace, 154-155
WWW-Authenticate header (HTTP response messages), 141

X–Y–Z

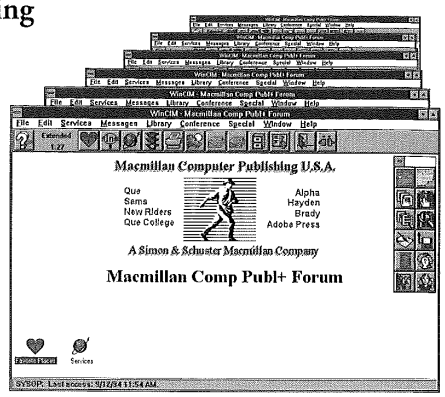
Yahoo directory (e-mail address), 161

IPR2025-00714
DATABRICKS EX1008 Page 429

GET CONNECTED
to the ultimate source of computer information!

The MCP Forum on CompuServe

Go online with the world's leading computer book publisher! Macmillan Computer Publishing offers everything you need for computer success!



Find the books that are right for you!
A complete online catalog, plus sample chapters and tables of contents give you an in-depth look at all our books. The best way to shop or browse!

- Get fast answers and technical support for MCP books and software
- Join discussion groups on major computer subjects
- Interact with our expert authors via e-mail and conferences
- Download software from our immense library:
  - Source code from books
  - Demos of hot software
  - The best shareware and freeware
  - Graphics files

Join now and get a free CompuServe Starter Kit!

To receive your free CompuServe Introductory Membership, call **1-800-848-8199** and ask for representative #597.

The Starter Kit includes:
- Personal ID number and password
- $15 credit on the system
- Subscription to *CompuServe Magazine*

MACMILLAN COMPUTER PUBLISHING

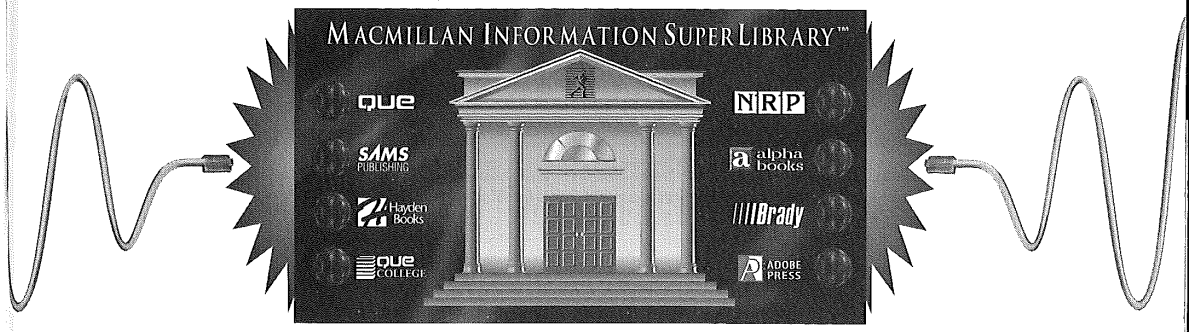
Once on the CompuServe System, type:

GO MACMILLAN

for the most computer information anywhere!

CompuServe

PLUG YOURSELF INTO...



THE MACMILLAN INFORMATION SUPERLIBRARY™

Free information and vast computer resources from the world's leading computer book publisher—online!

FIND THE BOOKS THAT ARE RIGHT FOR YOU!

A complete online catalog, plus sample chapters and tables of contents give you an in-depth look at ***all*** of our books, including hard-to-find titles. It's the best way to find the books you need!

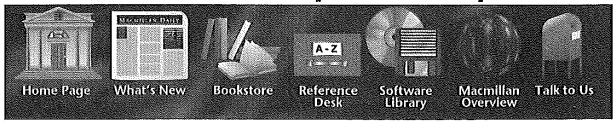
- STAY INFORMED with the latest computer industry news through our online newsletter, press releases, and customized Information SuperLibrary Reports.
- GET FAST ANSWERS to your questions about MCP books and software.
- VISIT our online bookstore for the latest information and editions!
- COMMUNICATE with our expert authors through e-mail and conferences.
- DOWNLOAD SOFTWARE from the immense MCP library:
  - Source code and files from MCP books
  - The best shareware, freeware, and demos
- DISCOVER HOT SPOTS on other parts of the Internet.
- WIN BOOKS in ongoing contests and giveaways!

TO PLUG INTO MCP:

GOPHER: gopher.mcp.com

FTP: ftp.mcp.com

WORLD WIDE WEB: **http://www.mcp.com**



WANT MORE INFORMATION?

CHECK OUT THESE RELATED TOPICS OR VISIT YOUR LOCAL BOOKSTORE

CAD

As the number one CAD publisher in the world, and as a Registered Publisher of Autodesk, New Riders Publishing provides unequaled content on this complex topic under the flagship *Inside AutoCAD.* Other titles include *AutoCAD for Beginners* and *New Riders' Reference Guide to AutoCAD Release 13.*

Networking

As the leading Novell NetWare publisher, New Riders Publishing delivers cutting-edge products for network professionals. We publish books for all levels of users, from those wanting to gain NetWare Certification, to those administering or installing a network. Leading books in this category include *Inside NetWare 3.12, Inside TCP/IP Second Edition, NetWare: The Professional Reference,* and *Managing the NetWare 3.x Server.*

Graphics and 3D Studio

New Riders provides readers with the most comprehensive product tutorials and references available for the graphics market. Best-sellers include *Inside Photoshop 3, 3D Studio IPAS Plug In Reference, KPT's Filters and Effects,* and *Inside 3D Studio.*

Internet and Communications

As one of the fastest growing publishers in the communications market, New Riders provides unparalleled information and detail on this ever-changing topic area. We publish international best-sellers such as *New Riders' Official Internet Yellow Pages, 2nd Edition,* a directory of over 10,000 listings of Internet sites and resources from around the world, as well as *VRML: Browsing and Building Cyberspace, Actually Useful Internet Security Techniques, Internet Firewalls and Network Security,* and *New Riders' Official World Wide Web Yellow Pages.*

Operating Systems

Expanding off our expertise in technical markets, and driven by the needs of the computing and business professional, New Riders offers comprehensive references for experienced and advanced users of today's most popular operating systems, including *Inside Windows 95, Inside Unix, Second Edition, Inside OS/2 Warp Version 3,* and *Building a Unix Internet Server.*

Orders/Customer Service **1-800-653-6156** Source Code **NRP95**

New Riders Publishing 201 West 103rd Street ◆ Indianapolis, Indiana 46290 USA

IPR2025-00714
DATABRICKS EX1008 Page 432

REGISTRATION CARD

Internet Agents: Spiders, Wanderers, Brokers, and 'Bots

Name _____ Title _____

Company _____ Type of business _____

Address _____

City/State/ZIP _____

Have you used these types of books before? ☐ yes ☐ no

If yes, which ones? _____

_____

How many computer books do you purchase each year? ☐ 1–5 ☐ 6 or more

How did you learn about this book? _____

Where did you purchase this book? _____

Which applications do you currently use? _____

_____

Which computer magazines do you subscribe to? _____

_____

What trade shows do you attend? _____

_____

Comments: _____

_____

_____

Would you like to be placed on our preferred mailing list? ☐ yes ☐ no

☐ **I would like to see my name in print!** You may use my name and quote me in future New Riders products and promotions. My daytime phone number is: _____

New Riders Publishing 201 West 103rd Street ◆ Indianapolis, Indiana 46290 USA

Fax to **317-581-4670** Orders/Customer Service **1-800-653-6156** Source Code **NRP95**

IPR2025-00714
DATABRICKS EX1008 Page 433

Fold Here

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 9918 INDIANAPOLIS IN

POSTAGE WILL BE PAID BY THE ADDRESSEE

NEW RIDERS PUBLISHING
201 W 103RD ST
INDIANAPOLIS IN 46290-9058

Internet Agents
Spiders, Wanderers, Brokers, and Bots

This is *the* reference guide to one of the most exciting technologies on the Internet today! Not only does this book explain the important concepts of agents, but it also examines the benefits of specific agents on the Internet.

With *Internet Agents: Spiders, Wanderers, Brokers, and Bots*, you'll learn about the basic principles, operations, and internal architectures of the following Internet agents:

- Web-indexing spiders
- Web maintenance robots
- Web shoppers and bargain-finders
- Worms and viruses
- MUD agents and chatterbots

You'll also get up-to-date information on the latest developments in the following fundamental technologies:

- HyperText Transfer Protocol 1.0
- Robot Exclusion Standard
- Encryption
- Digital signatures
- Digital cash
- Web transaction security
- Internet payment services

Plus, you'll get a list of Web robots and spiders on the Internet, and find the source code listings for WebWalker, a Web maintenance robot, and WebShopper, a personal comparison-shopping agent.

Internet Agents: Spiders, Wanderers, Brokers, and Bots—your guide to becoming better informed and more productive on the Internet!

Fah-Chun Cheong consults with start-up companies around the San Francisco Bay Area in the application of agent technologies for electronic commerce on the World Wide Web and Internet.

Cheong founded Agent Computing, Inc. in 1994 with a vision to develop innovative application-specific agent technologies for the Internet. He graduated with a Ph.D. in computer science from the University of Michigan in 1992 with a thesis on the construction of a new agent-oriented programming language.

$32.00 USA / $43.95 CAN
£29.49 Net UK

Category: Communications/Online—Internet
User Level: Accomplished - Expert

New Riders

http://www.mcp.com/newriders



IPR2025-00714
DATABRICKS EX1008 Page 435