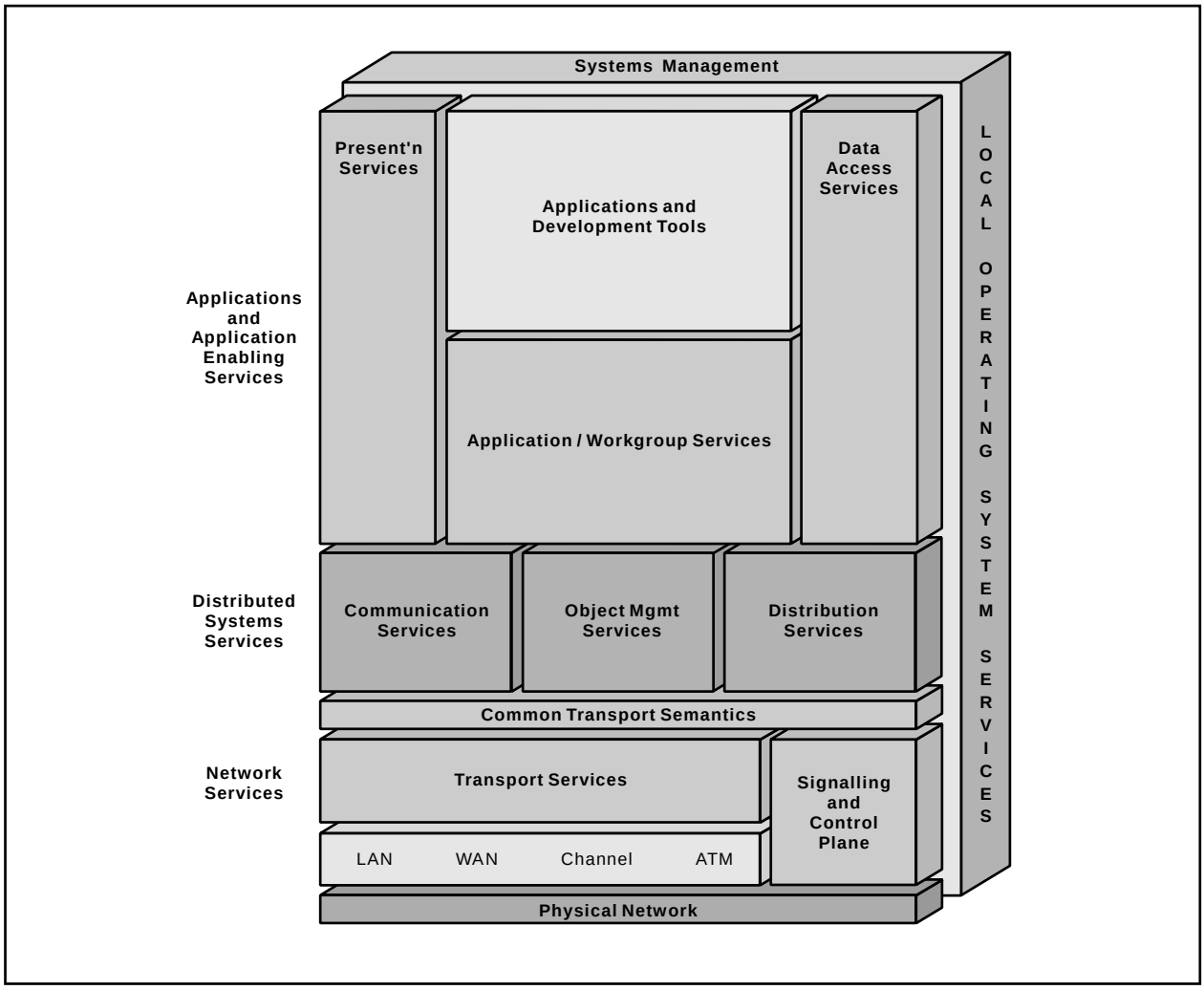
Open Blueprint

HTTP Resource Manager



G325-6593-00

Open Blueprint

HTTP Resource Manager

G325-6593-00

About This Paper

Open, distributed computing of all forms, including client/server and network computing, is the model that is driving the rapid evolution of information technology today. The Open Blueprint structure is IBM's industry-leading architectural framework for distributed computing in a multivendor, heterogeneous environment. This paper describes the HTTP resource manager component of the Open Blueprint and its relationships with other Open Blueprint components.

The Open Blueprint structure continues to accommodate advances in technology and incorporate emerging standards and protocols as information technology needs and capabilities evolve. For example, the structure now incorporates digital library, object-oriented and mobile technologies, and support for internet-enabled applications. Thus, this document is a snapshot at a particular point in time. The Open Blueprint structure will continue to evolve as new technologies emerge.

This paper is one in a series of papers available in the *Open Blueprint Technical Reference Library* collection, SBOF-8702 (hardcopy) or SK2T-2478 (CD-ROM). The intent of this technical library is to provide detailed information about each Open Blueprint component. The authors of these papers are the developers and designers directly responsible for the components, so you might observe differences in style, scope, and format between this paper and others.

Readers who are less familiar with a particular component can refer to the referenced materials to gain basic background knowledge not included in the papers. For a general technical overview of the Open Blueprint, see the *Open Blueprint Technical Overview*, GC23-3808.

Who Should Read This Paper

This paper is intended for audiences requiring technical detail about the HTTP Resource Manager in the Open Blueprint. These include:

- Customers who are planning technology or architecture investments
- Software vendors who are developing products to interoperate with other products that support the Open Blueprint
- Consultants and service providers who offer integration services to customers

Contents

Introduction

The Open Blueprint HTTP resource manager supports communication based on the HyperText Transfer Protocol (HTTP). HTTP enables Open Blueprint systems to participate in and communicate over the World Wide Web (WWW), the Internet, and enterprise intranets. HTTP-based communication is fundamental to the growth of the World Wide Web and the Internet.

HTTP support is often thought of as tightly coupled with Web browser support. Web browsers typically interact with the HTTP client to request the HTTP protocol services. However, in the Open Blueprint structure, the Web browser is defined as a separate resource manager because it requests functions using protocols other than HTTP, and has a broad set of presentation-oriented functions. (See the *Web Browser Resource Manager* component description paper.) Also, the HTTP resource manager can be accessed directly by other resource managers and applications such as editors.

The HTTP resource manager supports HTTP as a communications protocol for distributed, collaborative, hypermedia information systems. HTTP is an open, generic, protocol that can be used by various applications, such as name servers and distributed object management systems, through extension of its request methods. The HTTP resource manager supports the Common Gateway Interface (CGI) as the fundamental mechanism for invoking functions provided by server system applications and resource managers. The HTTP resource manager performs typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

The HTTP resource manager supports HTTP level 1.1 as defined by the Internet Engineering Task Force (IETF) HTTP Working Group. (See Appendix A, “References” on page 9, reference 1.)

Functions

Communications Support

The HTTP resource manager is activated by a request from an application, such as a Web browser. In HTTP, a request consists of a request line followed by header lines, possibly followed by a body. The request line consists of a method, a Uniform Resource Locator (URL), and the protocol name and version. An example of a request line is:

```
GET/index.html HTTP/1.0 User-Agent: Mozilla
```

In this example, the method is `GET`, which means this is a retrieve request. If a browser were submitting a form, the method would be a `POST`.

HTTP is a connection-oriented protocol. A *connection* is a transport layer virtual circuit established between two programs. In this case, the connection is between the HTTP resource manager client and server. The HTTP protocol defines the *message* as the basic unit of communication. A message consists of a structured sequence of bytes matching the syntax defined in the HTTP protocol specifications (See Appendix A, “References” on page 9, reference 2).

The objective of a communications connection is for the HTTP resource manager client to request a resource from the HTTP server. The resource is any network data object or service that can be identified by a URL. Resources can be available in multiple representations, such as multiple languages, data formats, size, resolutions, and so on. A resource can have one or more representation(s) associated with it at any given instant. Each of these representations is called a *variant*.

The information transferred as the "payload" of a request or response is known as an *entity*. An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body, as described in Reference 2.

Responses to requests typically consist of HTML, Multipurpose Internet Mail Extension (MIME) and, increasingly, Virtual Reality Markup Language (VRML) data types.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used.

Persistent Connections

A significant difference between HTTP 1.1 and earlier versions of HTTP is that persistent connections are the default behavior of any HTTP connection. Unless otherwise indicated, the client can assume that the server will maintain a persistent connection. Without persistent connections, a separate TCP connection is established to fetch each URL, increasing the load on HTTP servers and causing congestion on the Internet. The use of inline images and other associated data can require a client to make multiple requests of the same server in a short amount of time.

Persistent HTTP connections have a number of advantages:

- By opening and closing fewer TCP connections, processing time is saved, and memory used for TCP protocol control blocks is also saved.
- HTTP requests and responses can be pipelined on a connection. *Pipelining* allows a client to make multiple requests without waiting for each response. A server sends its responses to those requests in the same order that the requests were received. Pipelining improves the performance of the communication between HTTP clients and servers.
- Network congestion is eased by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.

Content and Variance Negotiation

Most HTTP responses include an entity that contains information for interpretation by a human user. Naturally, it is desirable to supply the user with the "best available" entity corresponding to the request. For example, it is desirable to supply the text of a Web page in a language the user understands. Unfortunately for HTTP servers and caches, not all users have the same preferences for what is “best,”, and not all client systems are equally capable of rendering all entity types. For that reason, HTTP provides several mechanisms that enable *content negotiation* which is the process of selecting the best representation for a given response when there are multiple representations available.

Upgrade Support

A client can advertise its desire to use another protocol, such as a later version of HTTP, even though the current request has been made using HTTP 1.1. This eases the difficult transition between incompatible protocols by allowing the client to initiate a request in the more commonly supported protocol while indicating to the server that it would like to use a “better” protocol if available (where better is determined by the server, possibly according to the nature of the method or resource being requested.)

Multiple Web Site Support

The HTTP resource manager supports multiple Web sites using a single IP address. This support greatly simplifies large operational Web servers, where allocation of many IP addresses to a single host creates serious problems.

Client Functions

The HTTP resource manager client is a program that establishes connections to send requests to an HTTP resource manager server. The requester, known as a *user agent* in HTTP architecture terminology, is most often a Web browser, but can be an editor, a spider (a Web-traversing robot), another application, or another resource manager.

The HTTP resource manager client:

- Accepts requests from, for example the Web browser
- Uses the URL in the request to establish a connection with the appropriate HTTP server
- Can request that only part (a byte range) of the response entity be included within the response
- Supports the pipelining of requests over a persistent connection
- Delivers the response from the HTTP server part to the requester
- Caches the response from the HTTP server

Server Functions

The HTTP resource manager server is a program that accepts connections to service requests and sends back responses. The basic functional support is for origin server systems. An *origin server* is a server on which a given resource resides or is to be created. The HTTP resource manager server in an origin server:

- Uses the local file system API to access the requested documents.
- Acts as a repository for hypermedia document resources (such as home pages) defined using HTML.
- Uses the HTTP transport protocol to serve requests from the Web browser (using the HTTP client).
- Supports HTTP Basic Authentication, SHTTP, and HTTPS, which are described in “HTTP Security” on page 6.
- Provides application interfaces using the CGI between the HTTP server and another application or resource manager. (see Appendix A, “References” on page 9, reference 3, for additional information about CGI.) The HTTP server executes CGI-BIN scripts specified within the URL. Examples of key functions available through the CGI-BIN scripts are the invocation of search engines, programs that can dynamically create HTML pages from other content, and support for clicking on image maps. Many Open Blueprint resource managers such as Relational Database and Transaction Monitor can be invoked through the use of CGI-BIN scripts.

In addition to supporting origin servers, the HTTP resource manager supports proxy and gateway server configurations.

Proxy Server: A proxy server contains both the client and server components of the HTTP resource manager, enabling it to make requests on behalf of other client systems. Requests are serviced internally through caching, or by passing them on, with possible translation, to other server systems. A proxy server might not be able to access server sites that require authentication. Also, Web browser helper applications on the client might not be able to communicate through a proxy server to the origin server. Proxy server configurations are often used in conjunction with *firewalls* to control access to Internet and intranet resources. See the Security resource manager component description paper for information on firewalls.

Gateway Server: A gateway server system acts as an intermediary for some other server system. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client might not be aware that it is communicating with a gateway.

Caching

HTTP is typically used in distributed information systems that can be spread across the world, such as the World Wide Web, where performance can be a primary concern. The use of response caches can significantly improve performance in these environments. The HTTP protocol supports several different caching approaches. The goal of caching in HTTP is to reduce the need to send requests and to reduce the need to send full responses. By reducing the need to send requests, the number of network round-trips required is reduced. An expiration mechanism is used for this purpose. By reducing the need to send full responses, the network bandwidth requirements are lowered. A validation mechanism is used for this purpose.

A cache is a program's local store of response messages and the function that controls its message storage, retrieval, and deletion. A cache stores cachable responses to reduce the response time and network bandwidth consumption for future, equivalent requests. Any client or server can include a cache. A response is cachable if it can be used in answering subsequent requests. Not all responses can be cached usefully, and some requests can contain modifiers which place special requirements on cache behavior. HTTP requirements for cache behavior and cachable responses are defined in the HTTP Specification (See Appendix A, “References” on page 9, reference 2). Even if a resource is cachable, there might be additional constraints on whether the cached copy can be used for a particular request.

Responses can be first-hand (not cached) or cached. A response is first-hand if it comes directly and without unnecessary delay from the origin server, perhaps through one or more proxies. A response is also first-hand if its validity has just been checked directly with the origin server.

A cache behaves in a semantically transparent manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response that it would receive if the request had been handled directly by the origin server. However, requirements for performance, availability, and disconnected operation require that the goal of symantic transparency be relaxed. The HTTP protocol allows origin servers, proxy servers, and clients to explicitly reduce symantic transparency when necessary, such as during times of network congestion or responses to requests from (now) disconnected clients.

There is a wide variety of architectures and configurations of caches and proxies currently being experimented with or deployed across the World Wide Web; these systems include national hierarchies of proxy caches to save transoceanic bandwidth, systems that broadcast or multicast cache entries, organizations that distribute subsets of cached data through CD-ROMs, and so on. HTTP systems are used in corporate intranets over high-bandwidth links, and for access using PDAs with low-power radio links and intermittent connectivity. A goal of HTTP is to support the wide diversity of configurations

already deployed while introducing protocol constructs that meet the needs of those who build Web applications that require high reliability and, failing that, at least reliable indications of failure.

Basic Operation/Flows

The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URL, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.

Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this can be accomplished through a single connection (v) between the user agent (UA) and the origin server (O).

```
request chain ------------------------>
UA -------------------v------------------- O
<---------------------- response chain
```

A more complicated situation occurs when one or more intermediaries are present in the request/response chain.

```
request chain -------------------------------------->
UA -----v----- A -----v----- B -----v----- C -----v----- O
<------------------------------------ response chain
```

The figure above shows three intermediaries (A, B, and C) between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections. This distinction is important because some HTTP communication options might apply only to the connection with the nearest neighbor, only to the end-points of the chain, or to all connections along the chain. Although the diagram is linear, each participant can be engaged in multiple, simultaneous communications. For example, B can be receiving requests from many clients other than A, and/or forwarding requests to servers other than C, at the same time that it is handling A's request.

Any party to the communication can employ an internal cache for handling requests. The effect of a cache is that the request/response chain is shortened if one of the participants along the chain has a cached response applicable to that request.

Additional Server APIs

CGI is the industry standard programming API for HTTP servers. It is a simple and flexible API. However, CGI design was not optimized for high performance; every CGI invocation involves the creation and deletion of a thread and a TCP/IP socket. Because of this, CGI programs spend a disproportional amount of time in thread creation, initialization, and deletion. A series of enhanced APIs are being defined and offered by various vendors as candidates to supplement CGI. The Netscape Server API, ISI API and EGI all address the improvement of scalability and performance of CGI. However, none of these APIs is, by itself, expected to entirely replace CGI.

Netscape Server API The Netscape Server API is a cross-platform API designed to improve the performance of long running HTTP server programs.

ISI API The ISI API is designed to improve the performance and scalability of HTTP server programs. It leverages many Windows conventions and is available only on Windows platforms.

EGI The Enhanced Gateway Interface (EGI) is an IBM API designed to improve performance of long running HTTP server programs. EGI preserves the semantics of CGI where possible. It is straightforward to convert a CGI program to an EGI program with the modification of a few lines of code. EGI is designed to work with "trusted" HTTP server applications.

GOSERVE API The GoServe API defined by IBM allows HTTP to call a REXX program and allows that program to execute. The program has access to many server variables and commands, and can, if so designed, perform all of the server function itself. Examples of usage are the support of user pre-exits that can perform additional authentication and support of post-exits that can filter data.

WINCGI WINCGI is a simple mapping of the CGI interface that enables Visual Basic and Visual C++ programs to do "CGI like" functionality in a Window's environment (message based versus CGI procedural model).

HTTP Security

The HTTP resource manager supports three types of security:

- HTTP Basic Authentication
- Secure HyperText Transfer Protocol (SHTTP)
- HTTP over Secure Sockets Layer (HTTPS)

HTTP provides a simple challenge-response authentication mechanism which can be used by a server to challenge a client request and by a client to provide authentication information.

Basic Authentication Scheme

The basic authentication scheme is based on the model that the user agent must authenticate itself with a user-ID and a password for each realm.[1] The realm value is an opaque string that can only be compared for equality with other realms on that server. The server services the request only if it can validate the user-ID and password for the realm specified in the request URL.

The basic authentication scheme alone is not a secure method of user authentication, nor does it in any way protect the entity. The most serious flaw in basic authentication is that it results in the clear text transmission of the user's password over the physical network. If meaningful security is desired, basic authentication should be used over an SHTTP or HTTPS connection.

A common use of basic authentication is for identification purposes. For example, the user might be required to provide a user name and password as a means of identification to permit gathering of information such as accurate usage statistics on a server.

Additional Authentication Schemes

An HTTP server can return multiple challenges with a 401 (Authenticate) response, and each challenge can result in a different scheme being used by the client. It is possible to then use either SHTTP or HTTPS as a secure protocol variant. Alternatively, the SHTTP or HTTPS scheme name can be specified in the requesting URL.

SHTTP (or Secure HTTP) is a standard developed by the CommerceNet consortium to address the lack of security in the World Wide Web. SHTTP is supported by the HTTP resource manager for migration purposes, but it is being supplanted in the industry by HTTPS. HTTPS is a defacto standard developed

by Netscape for making HTTP flows secure. Technically, it the use of HTTP over the Secure Socket Layer (SSL).

Relationships to other Resource Managers

The HTTP resource manager is central to the use of the Internet and is a cornerstone of intranet technology. It provides communication support for the Web Browser resource manager. It provides the CGI interface to link Web browser users to server applications and resource managers, both directly and indirectly, for example through the Messaging and Queuing resource manager.

The HTTP resource manager uses Network Services (most often TCP/IP transport services) to access the physical network. HTTP uses the TCP/IP Secure Sockets Library to support secure connections.

[1] A realm is this context consists of a series of Web pages that can be accessed with the same user ID and password.

Appendix A. References

Reference 1: For information about the IETF HTTP Working Group, see URL:
http://www.ics.uci.edu/pub/ietf/http

Reference 2: For information about HTTP specifications, see URL:
http://www.ics.uci.edu/pub/ietf/http/draft-ietf-http-v11-spec-07.txt

Reference 3: For information about the CGI specification, see URL: http:// www.w3.org

Appendix B. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

APPN
IBM
IBMLink
IIN
Open Blueprint
OS/2

The following terms are trademarks or service marks of other companies:

X/Open	X/Open Company Limited in the U.K. and other countries

Appendix C. Communicating Your Comments to IBM

If you especially like or dislike anything about this paper, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply. Feel free to comment on specific error or omissions, accuracy, organization, subject matter, or completeness of this paper.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

- If you prefer to send comments by FAX, use this number:

  United States and Canada: 1-800-227-5088.

- If you prefer to send comments electronically, use one of these ID's:
  - Internet: **USIB2HPD@VNET.IBM.COM**
  - IBM Mail Exchange: **USIB2HPD at IBMMAIL**
  - IBMLink: **CIBMORCF at RALVM13**

Make sure to include the following in your note:

- Title of this paper
- Page number or topic to which your comment applies

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

G325-6593-00