ALCS World Wide Web Server
User Guide

October 13, 2006

ALCS Customer Support
Mailpoint 213
IBM UK Ltd.
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

Second Edition (October 2006)

This edition applies to Release 3, Modification Level 1, of Airline Control System Version 2, Program Number 5695-068, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments appears at the back of this publication. If the form has been removed, address your comments to:

ALCS Customer Support
Mailpoint 213
IBM UK Ltd.
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

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

Contents

Figures

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. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights 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:

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 830A
522 South Road
Mail Drop P131
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and Service Marks

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

ACF/VTAM	ES/9000	OpenEdition	OS/390
SP	WebExplorer	AIX	AT
IBM	MVS/ESA	OS/2	RACF
VTAM	z/OS		

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks of others.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other contries:

The following terms are trade marks of other companies:

Adobe and PostScript are registered trademarks of Adobe Systems Incorporated.

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows, and Internet Explorer are trademarks or registered trademarks of Microsoft Corporation.

NCSA is a registered trademark of the National Computer Security Association.

Netscape and Navigator are trademarks of Netscape Communications Corporation.

POSIX is a trademark of Institute of Electrical and Electronic Engineers.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Yahoo! is a trademark of Yahoo! Inc.

Other trademarks are trademarks of their respective companies.

Revision Information

Changes to this document since the First Edition (June 1997) are indicated with a revision bar like the one on this paragraph.

Special 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. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

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 enquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about nonIBM (VENDOR) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therfore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

Chapter 1. Introduction

This book provides an introduction to the functions of the Airline Control System (ALCS) World Wide Web server (or just “the ALCS Web server”). It also explains how you use the ALCS Web server.

1.1 What is a Web server?

This section gives a brief outline of some background concepts such as Web servers and browsers, the Internet, and intranets. If you already know about these things then you can skip this section.

1.1.1 Servers, clients, and browsers

A **Web server** is a program that provides services to other programs called **Web clients**. Figure 1 shows this schematically.

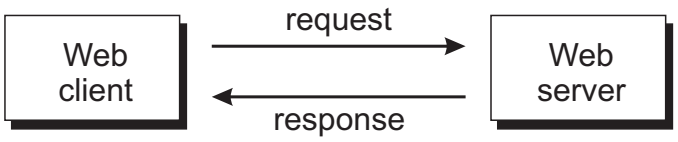


Figure 1. Web client and server

The most common type of service that a Web server provides is providing information to Web clients. We describe another type of service that Web servers provide in 1.1.11, “Interaction – forms” on page 9.

The most common type of Web client is a program called a **Web browser**. A Web browser is a program that allows a person to access information from Web servers. It does two important things for a person who wants information:

- It provides a simple way to ask for the information.
- It presents the information to the person in a form suitable for humans. (We were going to say “shows the information”, but Web browsers can also present sounds such as speech and music).

Usually (but not always) Web servers run on powerful computers such as IBM System/390s or Risk System/6000s. This allows one server to provide information to many browsers. Web browsers usually (but not always) run on much smaller computers such as PCs.

Web clients that are not actually Web browsers behave exactly like Web browsers except that they do not provide an interface for a human user. An example is a **gateway** that forwards requests from a client to another server and forwards that servers response back to the client. A gateway acts as both a server and a client. Figure 2 on page 2 shows this schematically.

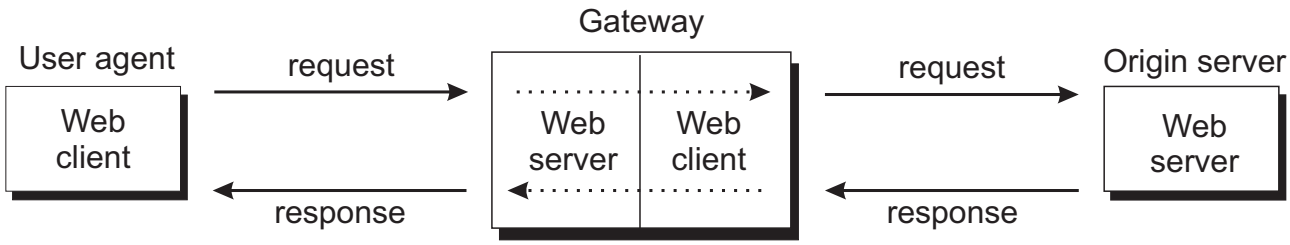


Figure 2. A gateway acting as a server and a client

In Figure 2 we show a single gateway between the real request originator, called the **user agent**, and the real destination for the request, called the **origin server**. There can be several gateways (and other exotic systems) between the user agent and the origin server.

Also the user agent may not be a Web browser. It could, for example, be a specialized computer program that systematically checks many Web servers to find out what information they hold. These specialized programs are often called **spiders** (because they "crawl around" on the Web), or more generally **robots** (because they are "machines" rather than humans).

The fact that there can be Web clients which are not Web browsers does not usually matter much to Web servers. In this book we mostly just talk about Web browsers and Web servers and only mention more exotic clients and servers when they make a difference to your ALCS Web server.

1.1.2 TCP/IP, HTTP, and HTML

Web servers and Web browsers talk to each other across a **transmission control protocol/Internet protocol (TCP/IP)** communication network. The TCP/IP protocol defines how a program:

- Establishes a communication link (called a connection) with another program.
- Transmits and receives data (bytes) across the link.
- Breaks the connection when the conversation is complete.

TCP/IP does *not* define such things as which of the programs establishes or breaks connections, or the format and meaning of the data that flows across the connection. Higher level protocols (sometimes called "layers") define these things.

Web servers and browsers use a higher level protocol called **hypertext transmission protocol (HTTP)** to define how they use TCP/IP. HTTP defines, amongst other things:

- Who establishes the connection (in this case, the browser).
- How the browser specifies what information it wants (the request format).
- How the server returns the information (the response format).
- Who breaks the connection (in this case, the server).

We have more to say about HTTP in 1.1.5, "HTTP – a little more detail" on page 6.

HTTP does *not* define the format or meaning of the information that the server transmits to the browser. HTTP 0.9 and 1.0 allow the server to send more or less any type of information including text, images, sound, programs, binary data, and

so on. HTTP 1.1 also allows the *browser* to send more or less any type of information.

Web browsers can “understand” various types of information. That is, they can display text and images, replay sounds, show movies, and so on. Exactly which types a browser can understand depends on the browser (and on other programs that the browser uses, called “viewers”). But all browsers can understand a special type of information called **hypertext markup language (HTML)**.

HTML consists of ASCII characters which is a mix of plain text and special “tags”. The tags tell the Web browser how to format and display the text (and they tell it other things too). For example, a piece of HTML information might contain a sequence something like this:

```
<p>This is
an <em>example</em> of some HTML
information.
It is a very simple example.
```

The <p> indicates the start of a new paragraph, the indicates the start of emphasized text, and the indicates the end of emphasized text. A browser might display the example something like this:

This is an *example* of some HTML information. It is a very simple example.

HTML tags can do much more complex things, such as including pictures, arranging text (and other things) into tables, and so on. We have more to say about it in 1.1.6, “HTML – a little more detail” on page 6, 1.1.7, “Imbedded objects” on page 7, and 1.1.8, “Hyperlinks” on page 8.

To summarize, a typical Web transaction comprises:

1. A person wants some information, they tell the Web browser what they want (see 1.1.4, “URLs” on page 5).
2. The Web browser connects to the appropriate Web server (TCP/IP).
3. The browser sends (TCP/IP) a request for the information (HTTP).
4. The Web server sends (TCP/IP) a package of data (HTTP) containing the information (HTML) back to the Web browser.
5. The Web server breaks the connection (TCP/IP).
6. The Web browser presents the information to the person.

Figure 3 on page 4 shows this dialog figuratively.

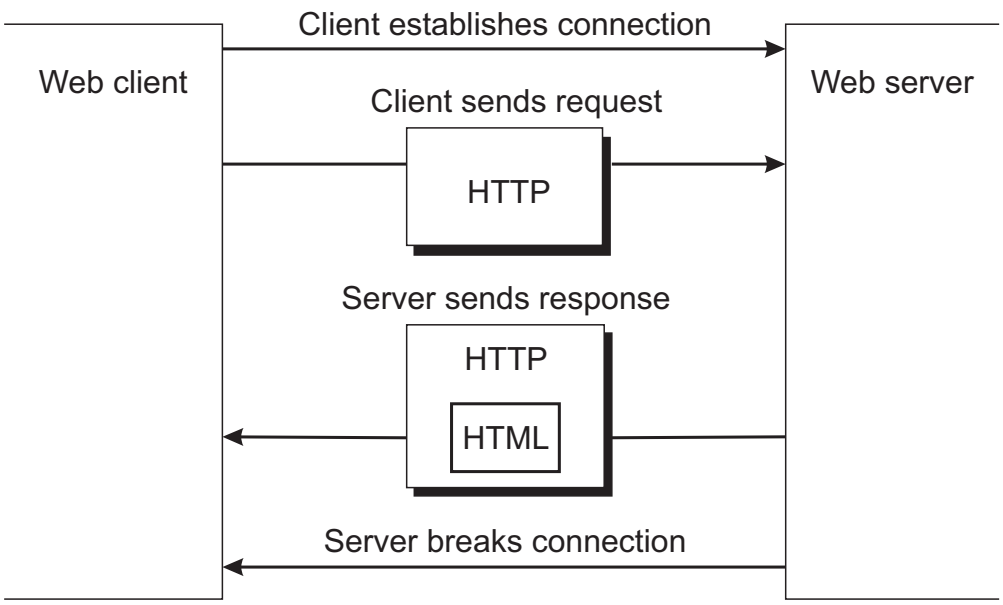


Figure 3. A typical Web request/response pair

Note

The above discussion assumes that the two programs which are communicating really are a Web browser (or other Web client) and a Web server. This is not necessarily true.

Any TCP/IP program can establish a connection with a Web server. The program can then transmit valid HTTP requests to the server (in which case the server responds appropriately), or anything else, including meaningless rubbish (in which case the server responds with an HTTP error message or breaks the connection or both).

Similarly, Web browsers can and do connect to TCP/IP programs that are not Web servers.

1.1.3 IP address, host names, and ports

If you want to attach a computer to an IP network such as the Internet or a private intranet, it must have a unique **IP address**. The Internet Network Information Center (InterNIC) is responsible for allocating IP addresses for computers attached to the Internet.

An IP address is a 4-byte binary number. The usual way to represent an IP address is the **dotted decimal notation**. The dotted decimal notation represents the contents of each byte as a decimal number and separates the four decimal numbers with dot (.) characters, so that a typical IP address might be `'129.34.139.30'`.

For most purposes (especially in URLs, see 1.1.4, “URLs” on page 5), you do not specify IP addresses. Instead, you use a **host name** (also called an “Internet domain name”). The InterNIC is responsible for allocating Internet domain names for computers attached to the Internet.

Host names are easy to remember character strings (well, easier than IP addresses), usually comprising several parts separated by dots. A typical host name might be 'www.ibm.com'.

A single computer often runs a number of different types of TCP/IP server, each using its own protocol. You select which server you want to connect to by specifying a **port**. In theory, you can associate any server with any port, but in practice there are **well known ports** that people expect you to use for particular types of server.

Web servers (using HTTP) are usually associated with port 80, but if you have more than one Web server on the same computer, each must have its own unique port. Typically you use port 80 for your “main” server and use other ports (often 8008 or 8080) for other Web servers.

1.1.4 URLs

When you use a Web browser, you must tell the browser what information you want. You do this by providing a **uniform resource locator (URL)**. If you want to get information from a Web server (there are other sorts of server), you provide a URL that contains the following:

- The fact that it's a Web server (rather than some other sort of server)
- The “name” of the Web server that holds the information (the host name)
- The port that the server uses
- The “name” of the piece of information that you want (the file name or “path”).

Your URL might look something like this:

```
http://www.someplace.com:80/interesting/information.html
```

`http`	means you are using HTTP (to talk to a Web server).
`www.someplace.com`	is the host name of the Web server.
`80`	is the port that the server is using.
`interesting/information.html`	is the “name” of the information.

In practice, you can usually omit the port from a URL. An HTTP URL “assumes” that the server is using port 80 unless you say different; you could use the following URL instead of the example we show above:

```
http://www.someplace.com/interesting/information.html
```

The full syntax of an HTTP URL is slightly more complex than this. We describe it in Appendix C, “HTTP uniform resource locators” on page 117.

Most of the time you are using a Web browser, you don't need to type in (or even know) the URL for the next piece of information you want. That's because of a clever feature of HTML, called hyperlinks. We have more to say about them in 1.1.8, “Hyperlinks” on page 8.

1.1.5 HTTP – a little more detail

As we mentioned in 1.1.2, “TCP/IP, HTTP, and HTML” on page 2, HTTP defines (amongst other things) the format of the requests that Web browsers send to servers and the format of the responses that Web servers send to browsers.

There are now (at the time we are writing this) three versions of HTTP:

0.9 An old and very simple HTTP, still used by some old browsers and servers.
1.0 A more advanced HTTP, used by most browsers and servers.
1.1 A new HTTP, used by some new browsers and servers.

The main differences between these HTTP versions are:

Headers HTTP 0.9 requests (from the browser to the server) are very simple. All they do is pass the URL and say “get me the information”. HTTP 1.0 and later versions have a **request header** that includes information about the Web browser – and about the user of the browser – in the request header. Web servers can use this information to customize responses.

HTTP 0.9 responses (from the server to the browser) are also very simple. They consist of the information that the browser requested (usually HTML). HTTP 1.0 and later versions have a **response header** that contains information about the server, and **metainformation**. Metainformation is information about the information; for example, it says what the information is (HTML, an image, a movie clip, and so on).

Methods HTTP 0.9 only allows the browser to get information from the server. HTTP 1.0 and later versions allow more types of request. For example, they allow the browser to send information to the server. These “types of request” are called **methods**.

Usually any browser or server that can use a particular version of HTTP can also use any older version. For example, an HTTP 1.1 server can usually communicate with an HTTP 1.0 or HTTP 0.9 browser. and an HTTP 1.1 browser can usually communicate with an HTTP 1.0 or HTTP 0.9 server.

1.1.6 HTML – a little more detail

HTML contains text together with tags. The tags tell the Web browser how to format and display the text; for example there are HTML tags that say things like:

“Start a new paragraph”
“Use an Italic font for this text”
“Arrange this text into a table”
and so on.

Earlier, we showed an example fragment of HTML like this:

```
<p>This is
an <em>example</em> of some HTML
information.
It is a very simple example.
```

and explained that a Web browser might display it like this:

This is an *example* of some HTML information. It is a very simple example.

Note that the browser itself decides on some aspects of formatting. In this example, the browser decides which fonts to use and how long to make each line of text. This allows the browser to present information in a way that best suits the computer it is running on. For example, a Web browser can use longer lines (and more of them) on a larger display.

When you use a Web browser, you usually start off with a URL that refers to a piece of HTML. A piece of HTML is called a **Web page**. A Web page does not correspond to a conventional page of text, like a page of this book; this entire book could be a single Web page.

Web browsers allow you to scroll backwards and forwards through a page that is too big to display all at once in the screen. Similarly they allow you to scroll left and right across a page that is too wide; but most browsers try to adjust the text line length and so on so that you do not need to scroll left and right.

1.1.7 Imbedded objects

In addition to text formatting tags, HTML provides tags for including images, movies, sound, and so on in Web pages. A tag to include an image might look like this:

```
<img src="picture.jpeg" alt="Sorry you can't see my nice picture">
```

This tag means:

> Put an image (`img`) here. To get the image, ask the Web server for the file `picture.jpeg`. If you can not (or do not want to) display pictures, display the text “Sorry you can't see my nice picture” instead.

If the browser is displaying images, it sends a *separate* request to the server, asking the server to send the image file.

This image is an example of an **imbedded object**. A single Web page can contain many imbedded objects. Most browsers send a request to the server for each imbedded object as soon as the HTML tag arrives – they do not wait for a complete object to arrive before they request another.

In our example, we tell the browser to request the image from the same server that provided the HTML page. But we *could* tell the browser to get the image from another server by including the full URL for the image; something like:

```
<img src="http://another.server/picture.jpeg"
alt="Sorry you can't see someone else's nice picture">
```

Note that the person using the browser does not need to know the URL for the imbedded object. And the user does not need to know that different components of the Web page come from different servers.

1.1.8 Hyperlinks

In addition to references to imbedded objects (which are part of the same Web page), HTML allows a Web page to contain references to other Web pages. These references are called **hyperlinks**.

Browsers display hyperlinks as a piece of text (often in a distinctive color) or an image (often with a distinctive border). The user selects the text or image – typically using a mouse to “point and click” – and the browsers requests the corresponding Web page.

Some HTML that includes a hyperlink might look like this:

```
<p>There is lots of information about
<a href="http://some.server/cricket.html">cricket</a>
available on the Web.
```

The browser might display something like this:

There is lots of information about **cricket** available on the Web.

If you “point and click” to select the word “cricket” then the browser requests the page `http://some.server/cricket.html`.

Note that, as with imbedded objects, the person using the browser does not need to know the URL for the Web page.

1.1.9 Web sites

If you want to provide information (and other services) to people with Web browsers, you will normally need many Web pages. Collectively, these Web pages are called a **Web site**.

You want to make it as easy as possible for people to find the page or pages they want. The usual way to do this is to provide a simple URL that requests a **welcome page** (sometimes called a **home page**).

The home page contains a description of who you are and what you offer, together with hyperlinks (often arranged as a list) to other pages. In this way, your home page forms a “table of contents” for your other pages. Of course, the pages that your home page links to can themselves be lists of hyperlinks.

For example, a sporting goods manufacturer might have a home page that lists particular sports; golf, cricket, synchronized swimming, and so on.

Selecting one of these sports might request a page that lists items related to that sport. Selecting “cricket” might display a list including pads, stumps, bales, and so on. Selecting one of these items might request a page giving more detailed information about that particular item; listing the available styles, sizes, prices, and so on.

Of course, the list you get when you select “cricket” might include a hyperlink to a movie clip that shows why cricket is so fascinating, together with a clip of the mellow sound of leather on willow.

Note that a Web site does not need to be a single Web server. Different pages connect to each other using hyperlinks so that different pages can be on different servers. This allows an organization to distribute the work-load across several servers or to use specialized servers for particular pages. It is also relatively easy to move pages from one server to another.

Because hyperlinks “hide” the complexity of the URLs, having multiple servers does not make a Web site harder to use – in fact users will probably not even notice that there is more than one server.

1.1.10 Static and dynamic Web pages

As we have explained, a URL can request a Web page. In the simple case, the URL refers to a piece of HTML that the Web server keeps on a data base. This is sometimes called a **static** page (even if it includes movie clips!).

But a URL can also refer to a *program* that runs in the Web server. The program decides what information (Web page) to return. This is sometimes called a **dynamic** page.

These programs are often called “scripts”. The sorts of things they can do include:

- Generate (or select) a page that is customized for a particular browser.
- Collect information from a variety of data bases (and other sources) and present that information on a single page.
- Include up-to-the minute information – for example the exact number of seats available on an airline flight.
- Vary the content of a page so that each time a person requests it they get new and interesting information.

1.1.11 Interaction – forms

The Web pages we have described so far present information to the browser. There is a special type of Web page, called a **form** that contains input fields. Forms allow more sophisticated interaction between the browser and server than simple hyperlinks. A hyperlink is a simple request to the server (“show me this page”). But a form can create a much more complex request.

For example, an airline Web page could provide a menu of hyperlinks showing schedules for a selection of routes. Alternatively, a form could allow the user to select particular departure and arrival cities (or airports), a particular date, preferred seating class, and so on. The server can then display customized schedule information.

Forms can also allow business transactions. For example, an airline could allow customers with browsers to make seat reservations. The airline Web server provides forms so that the customer can enter details such as which flight they require, their name, address, telephone number, and so on.

As you would expect, forms generate URLs that are more complex than simple hyperlinks (the input information is appended to the URL). And these URLs normally refer to programs (scripts). The program can access the information provided with the URL. Figure 4 on page 10 shows a typical forms dialog figuratively.

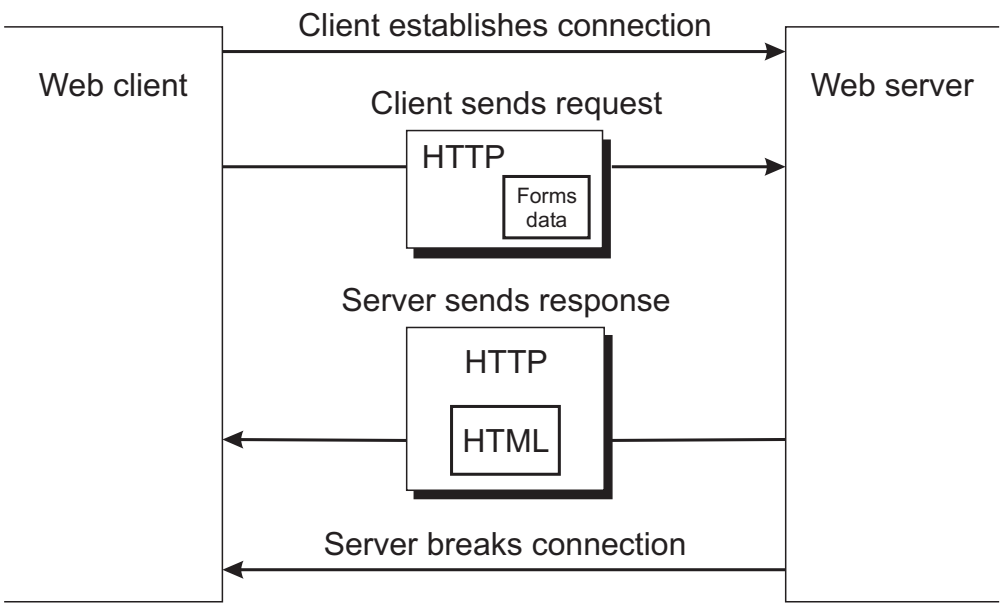


Figure 4. A typical Web forms request/response pair

1.2 The Internet and the World Wide Web

The Internet is a large set of interconnected but independent TCP/IP networks. The individual networks are owned and run by a variety of organizations, including:

- Commercial organizations
- Governments
- Educational institutions
- and so on.

In addition, a number of service providers allow private individuals to connect to the Internet.

In this way, the Internet connects an enormous number of computers together. They range from the smallest personal computers to the largest and most powerful server machines. Many of these computers run Web servers. Taken together, all these servers are called the **World Wide Web**, or just “the Web”. The World Wide Web provides a bewildering variety of information and other services to a huge number of potential users.

An enterprise may want to connect to the Word Wide Web for at least two reasons:

- To allow its staff access to the information and other services on the Web.
- To provide information and services to the Web (to have a Web site).

1.3 Intranets and client-server computing

In addition to allowing its staff to access the Web and having a Web site, an enterprise can use the same technology to provide a “private” Web containing Web servers that provide information and services exclusively for its own staff. This type of private Web is called an intranet.

Intranet technology can be a very effective way to provide corporate computing services. It is an example of **client-server** computing. Client-server computing provides computer services by dividing work between a client (usually single-user work-station, such as a personal computer) and a server (usually a large powerful computer, such as an IBM System/390).

Client-server computing allows the server to concentrate on the things it is good at, such as maintaining a large data base, while the client concentrates on the things it is good at, such as providing a “friendly” user interface.

Although there are clear advantages to a client-server strategy for corporate computing, some implementations introduce problems of their own. Some of these problems relate to maintaining fast-changing data in client computers, for example:

- Many transaction processing systems maintain inventory information; how many of an item are in stock, how many rooms in a hotel are unoccupied, how many seats are available on an airline flight, and so on.

  If this information is stored in client computers, it can be difficult to ensure that all clients are kept completely up-to-date.

- In a competitive market, enterprises often need to frequently update apparently “static” information, such as the range of products and services offered, prices (including special offers), and so on.

  Again, it can be difficult to keep this information up-to-date if it is maintained in client computers.

It is often more efficient and reliable to keep fast-changing information centrally in server computers, rather than attempting to keep up-to-date copies in a (possibly large) number of client computers.

Other problems relate to maintaining programs in client computers. It can be expensive and time consuming to upgrade programs in client computers. This can be a particular problem if:

- The client computers are installed over a wide geographic area
- The users of the client computers are not computing specialists
- Different users have different types of client computer, or run a different mix of programs, or both.

For certain types of client-server implementation, failure to keep all client computers' programs absolutely up-to-date can have serious consequences. In particular:

- If the client programs implement business processes that change.

  For example, a client program might include pricing algorithms. Any change to the way your enterprise calculates prices requires an immediate update to the program in all client computers.

- If changes to the server application require corresponding changes to the client programs.

  For example, a client program might depend on the format of messages from the server (this is particularly true for “screen scraper” programs). Any change to the server application must be exactly synchronized with the corresponding change to the program in all client computers.

Web technology avoids many of these problems. In particular:

- It maintains business-critical data and application function centrally. This ensures that all end-users have access to absolutely up-to-date information and services.
- It “decouples” the client and server programs so that each can be upgraded independently of the other.
- It does not constrain the choice of client platform. Web browsers are available for a wide variety of computers and operating systems. This means that your staff are not constrained to use one particular system, each can select one that best satisfies the requirements of their job.

Chapter 2. Overview of the ALCS Web server

This chapter gives a brief overview of the ALCS Web server.

2.1 Why an ALCS Web server?

There is a wide variety of Web servers available on a wide range of platforms. So why would you want another one? Why would you want to use ALCS as a Web server in your enterprise?

This section provides some answers to these questions.

The ALCS Web server is designed so that it can fulfil several roles:

- It can act as a general-purpose Web server.
- It can act as a special-purpose Internet Web server, exploiting your existing ALCS applications to offer services and advertising on the Internet.
- It can act as an intranet server, creating a “user friendly” interface to your existing applications for your staff.

2.1.1 ALCS as a general-purpose Web server

ALCS is a first class transaction processing platform. In many ways, this makes ALCS an ideal platform for a Web server. Consider:

- **ALCS is optimized for transactions that are very similar to Web transactions.**

  Typical Web transactions require fast access to a large data base of up-to-the-minute information and perform little or no complex processing of that data.

- **ALCS provides exceptionally high availability.**

  This is important for a Web server where end users are likely to turn to other providers if a particular site is unavailable when they need it.

- **ALCS is capable of handling high transaction rates.**

  This is important for a Web server where seasonal factors or special events can create surges in demand for the services.

- **If you already have an ALCS system, you may not need another Web server.**

  The ALCS Web server provides enough function to support a relatively sophisticated Web site. You may prefer to exploit ALCS's capabilities instead of introducing additional platforms that might require new computers, new programs, and additional staff.

  Of course, there is no reason why you should not use ALCS as a Web server even if you already have, or plan to have, other Web servers.

ALCS supports most of the functions that you might require from a general-purpose Web server. It supports Web pages that include text, images, audio and video clips, Java applets, and so on. It can also act as a repository for other kinds of objects, such as programs, that Web browsers can down-load.

ALCS also allows you to implement dynamic Web pages, using “conventional” ALCS application programs to select or create Web pages.

Note: It is generally easy to move Web pages from another Web server to ALCS, or from ALCS to another Web server. Web technology means that you do *not* need to commit yourself to using a particular server or platform.

2.1.2 ALCS as a special-purpose Internet Web server

The ALCS Web server is a simple way to provide information and services on the Internet, using your existing application functions. It is possible that your ALCS data base contains information that you could make available to the general public, and to specific clients, on the Internet. For example, an airline's ALCS data base might contain:

- Schedule information
- Fares information
- Seat availability on specific flights
- and so on.

By using ALCS as a Web server, it is easy to make this information available on the Internet. You do not need to maintain copies of this information on other data bases, or to ship the information from ALCS to another system – ALCS can access it directly.

Also, it is possible that your ALCS applications provide functions that you could make available to the general public, and to specific clients, on the Internet. For example, an airline's ALCS application might support:

- Flight reservations
- Seat selection
- Special services selection (special meals, and so on)
- Special options for frequent flyers
- Automatic cancellation for unconfirmed reservations
- and so on.

By using ALCS as a Web server, it is easy to make these functions available on the Internet. You do not need to duplicate the functions on another system – ALCS can access them directly.

2.1.3 ALCS as an intranet server

ALCS's Web server capabilities can be a flexible and inexpensive way to implement a “user friendly” interface to your applications for your own staff.

If you skipped the section 1.1, “What is a Web server?”, you might like to refer to 1.2, “The Internet and the World Wide Web” on page 10, which highlights some advantages of this approach to client-server computing.

2.2 What is the ALCS Web server?

ALCS includes a number of components that together comprise the Web server. They are:

- TCP/IP support
- HTTP support programs
- The access log

- The hierarchical file system
- The hierarchical file system application
- PC file transfer
- The HTTP application interface
- Installation-wide exits for the Web.

The following sections describe these components in more detail.

2.2.1 TCP/IP support

ALCS's TCP/IP support allows you to connect an ALCS system to a TCP/IP network. This includes connection to a “private” intranet, or to the “public” Internet, or both.

ALCS's TCP/IP support is general purpose. You can exploit it by developing your own applications that use the TCP/IP “verbs” directly. You could, for example, use ALCS's TCP/IP support to develop your own Web server. But that is unnecessary because ALCS already includes a Web server.

(You can also use ALCS's TCP/IP support as a transparent transport protocol. For example, if you have applications that use the ALCS router to communicate across LU6.1 links, you can choose to use TCP/IP instead without changing your applications.)

2.2.2 HTTP support programs

ALCS includes a suite of ECB-controlled programs that support the HTTP protocol. This suite of programs is the “heart” of ALCS's Web server. ALCS's HTTP support programs receive and decode requests from Web browsers.

If the request is for a “static” page, the ALCS HTTP support programs retrieve the requested page from the hierarchical file system (see 2.2.4, “The hierarchical file system” on page 16) and transmit it to the Web browser.

If the request is for a “dynamic” page, the ALCS HTTP support programs call your application programs to select a response page, or to build the response page (see 2.2.7, “The HTTP application interface” on page 17) and transmit it to the Web browser.

The ALCS HTTP support programs allow you to provide installation-wide exit programs that customize the ALCS Web server.

2.2.3 The access log

The ALCS HTTP support programs log Web accesses to the WWW real-time sequential file. The records on this file conform to the **common access log format**. You can browse this log to check who is accessing which of your Web pages (and other objects).

Alternatively, there are several programs available which can analyze common access log format files and generate reports.

2.2.4 The hierarchical file system

ALCS supports a hierarchical file system (HFS). You are probably familiar with the hierarchical file systems implemented in UNIX and PC platforms. The ALCS HFS is similar to the UNIX file system. A typical file in the ALCS HFS might have a name like:

```
/www/interesting/information.html
```

which refers to a file called `information.html` that is in a directory called `interesting` which is a subdirectory of `www`.

The ALCS Web server uses the ALCS HFS. In fact, in this release of ALCS, the HFS is primarily intended for use by the ALCS Web server.

As you might expect, when the ALCS Web server receives a URL of the form:

```
http://www.someplace.com/interesting/information.html
```

it interprets `interesting/information.html` as a file name within its part of the ALCS HFS. If the Web server directory is `/www` then the Web server retrieves the file:

```
/www/interesting/information.html
```

Files in the ALCS HFS are “unstructured” in the sense that a file contains an arbitrarily long sequence of bytes. The sequence of bytes is not divided into logical records.

An ALCS HFS file can contain data in one of the following forms:

EBCDIC The file is a string of EBCDIC characters
ASCII The file is a string of ASCII characters
BINARY The file is a string of bytes containing arbitrary data.

The ALCS Web server normally stores HTML in ASCII files and imbedded objects (such as images) in BINARY files.

You may be wondering how you create these files that the ALCS Web server uses. We discuss this in a little more detail in 2.2.5, “The hierarchical file system application” and 2.2.6, “PC file transfer” on page 17.

As we mentioned in 1.1.10, “Static and dynamic Web pages” on page 9, URLs can refer to programs (sometimes called scripts). The ALCS HFS allows you to make an HFS “file” that is actually an ECB-controlled program. When the Web server receives a corresponding URL, it calls the ECB-controlled program to process the request. We discuss this in a little more detail in 2.2.7, “The HTTP application interface” on page 17.

2.2.5 The hierarchical file system application

The ALCS HFS includes an application that allows you to manage the files (and programs) in the HFS. To use the HFS application, you route your terminal to the application with the ALCS `ZROUT` command.

The application supports a number of commands (input messages) that are designed to be easy to use if you are already familiar with UNIX, DOS, OS/2, and other platforms that have an HFS. For example the ALCS HFS application supports:

Command	Function
`dir`, `li`, `ls`	List the files in a directory.
`cd`, `chdir`	Change the current directory.
`md`, `mkdir`	Create (make) a directory.

and so on.

See Chapter 7, “HFS and PC file transfer commands” on page 79 for a full description of the ALCS HFS commands.

2.2.6 PC file transfer

Although ALCS is an excellent Web server, we do not expect you to use ALCS to create your Web pages. Instead, there is a wide range of tools available on PC platforms. These tools include:

- Graphics tools for creating and manipulating images
- Multimedia tools for creating and manipulating sound and movie clips
- WYSIWYG text editors for creating HTML text
- Web authoring tools for building Web sites
- and so on.

ALCS allows you to take advantage of these tools. You can use them to create files on a PC and then you can “up-load” the files to the ALCS HFS where the ALCS Web server can use them.

To up-load files from a PC to ALCS, you use the PC `SEND` command. (You may already use this command to up-load ALCS maintenance fixes from a PC to TSO.) 7.11, “Up-loading files from a PC” on page 84 describes the PC `SEND` command for up-loading files to ALCS.

2.2.7 The HTTP application interface

When the ALCS Web server receives a URL that the HFS identifies as an ECB-controlled program, it calls the ECB-controlled program with ENTDC.

It passes the following information to the program:

- A control block that contains information extracted from the original request. This information includes:
  - The HTTP version that the Web browser is using
  - The URL
  - Information about the browser
  - and so on
- If the request contains forms data (see 1.1.11, “Interaction – forms” on page 9), a control block that contains the forms data (names and values) in an easy-to-use tabular form.
- A record (called the HFS state control block) which the program can use to save information about the original request.

The ECB-controlled program can perform any appropriate application functions. For example, in an airline reservation system, the program might extract flight schedule information or make a seat reservation. The program then selects (or builds) a response object and send it to the browser. Typically this will be an HTML Web page.

Of course, the program can call (ENTER/BACK) other ECB-controlled programs, including your existing ALCS application programs, to help build the response object.

For HTTP versions above 0.9, the response object requires an HTTP header. The ALCS Web server includes a service program that builds this header.

Chapter 8, “HTTP application interfaces” on page 87 covers this topic in more detail.

2.2.8 Installation-wide exits for the Web

The ALCS Web server supports a number of installation-wide ECB-controlled exits that you can use to customize the behavior of the Web server.

For example, the Web server assumes some naming conventions for HFS files. Based on these assumptions, the Web server determines the content type of the file when it is constructing HTTP headers. If you want to use different naming conventions, or if you want to have content types that ALCS does not know about, you can provide an installation-wide exit program that extends or replaces the way the Web server determines the content type.

Chapter 9, “Installation-wide exits for the Web” on page 103 covers this topic in more detail.

2.3 Prerequisites for the ALCS Web server

Before you can use the ALCS Web server, you must:

- Install the high-level assembler
- Install the logical string assist processor feature
- Install the ALCS TCP/IP and Web server PTFs
- Install the TCP/IP for MVS product
- Install a TCP/IP network – including a connection to the Internet, if you want your server to provide Internet services.

You also need to run ALCS system and communications generations, see Appendix A, “ALCS generation parameters for the Web server” on page 113.

Chapter 3. Security characteristics of the ALCS Web server

Unless you take special precautions to prevent it (see 3.4, “Firewalls” on page 21 and 3.4.5, “Proxy servers and SOCKS servers” on page 28), as soon as you connect a Web server to a TCP/IP network then anyone connected to that TCP/IP network can access the information and services that the server provides.

With many general-purpose Web servers, you need to take special precautions to ensure that people who connect to the server can only access information and services that you intend to provide.

For example, a general-purpose Web server running on a UNIX or PC platform usually “shares” the HFS with other applications on the same platform. This means the HFS contains a wide variety of information, programs, and so on that are not related to the information and services you intend the Web server to provide. Some of the information may be confidential, and some of the programs may do things that compromise the security or integrity of your system.

With a general-purpose Web server, you must tell the server exactly which files it can send to a browser, which programs a browser can invoke, and so on.

Also, HTTP 1.1 supports browser requests that update the HFS. For example, it supports requests to add and delete HFS files. If you use a Web server that supports these requests, you need to be careful to control who uses them and how.

Because the ALCS Web server is a special-purpose server, some of these considerations do not apply. In particular:

- The ALCS Web server will only access files and programs within one directory (usually the `/www` directory) and its subdirectories. It rejects any URL that refers to any other directory.
- Unlike UNIX and PC platforms, ALCS does not keep programs in the HFS. A browser can not accidentally or deliberately invoke one of your application programs by knowing (or finding out) where the program is stored in the HFS. The ALCS HFS only allows access to programs that you explicitly add into the HFS.
- The ALCS Web server does not support browser requests that update the HFS.

In summary, the ALCS Web server only provides information and services that you explicitly tell it to provide.

Of course, this does not mean that you can completely ignore security. The following sections discuss some security considerations that you need to think about. They are not intended as a comprehensive guide to data processing security. If you have questions about the security of your ALCS system, including the implications of the ALCS Web server, then IBM can provide specialist advice.

If you have more general questions about TCP/IP security, then your local IBM representative can put you in touch with people who can provide specialist advice.

3.1 Intranet-only server considerations

If you intend to have an intranet-only server; that is, your server:

- Connects to a private TCP/IP network (your intranet)
- Only provides information and services to your own staff

then you may not need to worry much about security. You can set up your ALCS Web server to provide exactly the information and services that you want your staff to access. But you should still consider:

- Is the physical security of the equipment attached to your TCP/IP network adequate?

  You may need to be particularly careful if you allow access to your intranet from a public switched network (dial-up access).

- Is there any connection between your private TCP/IP network and the public Internet?

  If there is then you should consider protecting the connection with a firewall. If you do not have a firewall between the Internet and your intranet then Internet users may be able to access your Web server.

- Do you want to provide the same information and services to all your staff (that is, to all that have access to your TCP/IP network)?

  If you intend to provide different services and information to different staff then you should consider using a gateway.

For more information about firewalls, including gateways, see 3.4, “Firewalls” on page 21.

3.2 Internet-only server considerations

If you intend to have an Internet-only server; that is, your server:

- Connects to the public Internet
- Only provides information and services to the general public

then you may not need to worry much about security. You can set up your ALCS Web server to provide exactly the information and services that you want the general public to access. But you should still consider:

- Does your ALCS Web server provide an unintended connection between your private TCP/IP network and the public Internet?

  If it does then you should consider installing a firewall (see 3.4, “Firewalls” on page 21).

3.3 Mixed intranet and Internet server considerations

If you intend to have a server that connects to both your private intranet and the public Internet, then security is more complex. In particular:

- You should consider installing a network-level firewall to control access to your intranet from the public Internet.
- Unless you intend to provide the same services and information to both your own staff and the general public, you may need a full-function firewall.

For more information about firewalls see 3.4, “Firewalls” on page 21.

3.4 Firewalls

If there is a connection between two TCP/IP networks then it is usually possible for someone connected to one of the networks to access servers connected to the other network.

In the particular case where your private TCP/IP intranet connects to the public Internet, members of the public can accidentally or deliberately abuse the services on your intranet. (Of course, this does not just apply to your ALCS Web server. It applies equally to any server connected to your TCP/IP network.)

To prevent this, you can install a **firewall**. A firewall is a special computer system that connects two TCP/IP networks and controls the data that flows between the two networks. Figure 5 shows this schematically.

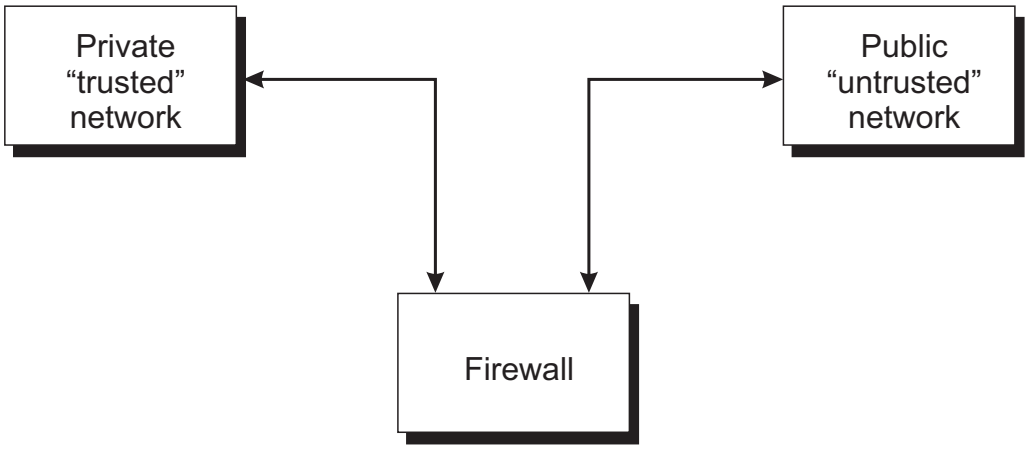


Figure 5. Firewall

The firewall is often a dedicated processor that only runs specialized security programs. An example is IBM's Firewall Technologies for MVS. Some firewalls consist of several processors which work cooperatively. Generally firewalls have (or should have) the following characteristics:

- All connections between the two networks are firewalls.

  If there are other connections that bypass the firewall or firewalls then there is not much point in having firewalls at all.

- The firewall's connections to the two networks are physically separate.

  This allows the firewall to “know” which of the two networks data comes from. Without this safeguard, someone in the untrusted network might send data with a false originator address so that the data appears to come from someone in the trusted network. This technique is called **spoofing**.

- The firewall itself is protected against accidental or deliberate interference.

  At the least, this means that only trusted staff can update the programs and data that control the firewall functions. One way to achieve this is to use separate dedicated machines for the firewall so that only authorized security staff have any access to the machine's programs and data.

- The firewall logs traffic that flows through it and (importantly) traffic that it rejects.

  This helps you to check that the firewall is protecting your network properly and to identify and investigate attempts to penetrate your network.

3.4.1 Network-level and application-level firewalls

A distinction is often made between network-level firewalls and application-level firewalls:

- **Network-level** firewalls, also called **screens** or **filters**, generally restrict who can talk to what.

  For example, you might configure a screen so that any data originating on the public network can only pass to certain specified addresses on your private network. You can also configure a screen to allow specified originators to route data to specified destination addresses (but be aware that it is possible for someone to send data with a fake originator address).

  Network-level firewalls can also “hide” the actual IP addresses that you use in your private TCP/IP network. That is, a system in your private network can have a real IP address (which systems in your private network use) and a different IP address that systems in the public Internet use. This is called network address translation (NAT). Reasons you might want this include:

  – It enhances security. It makes it more difficult for someone who finds a way to bypass your firewalls to access your private systems.

  – The IP addresses in your private network can be the same as other IP addresses on the Internet (or on other private TCP/IP networks).

- **Application-level** firewalls, also called **gateways** generally work together with network-level firewalls to restrict who can use what services.

  For example, you might configure a gateway Web server so that clients on the public Internet can only use a subset of the services (URLs) that clients on your private network can use. Note that a single gateway server can do this for a number of origin servers.

  Application-level gateways can also “hide” the organization of your origin servers. A user sees a single server (the gateway) which provides a consistent set of services even if you have many origin servers and sometimes move functions between them.

Note that network-level firewalls and application-level firewalls are firewall *functions*. It is often possible and convenient to run both functions in the same physical machine. Even if you use separate network-level and application-level machines, you normally connect them together to form a single firewall.

3.4.2 Example firewall configurations

This section discusses a number of example configurations that include an ALCS Web server. For each example, we explain the type of firewall you might need, and explain what the firewall does.

IBM markets a range of firewalls. If you want more information, contact your local IBM representative.

ALCS serving a single network – no firewall

If you have a private TCP/IP network that does not connect to the public Internet, you can use the ALCS Web server to provide information and services with no firewall. Figure 6 shows this.

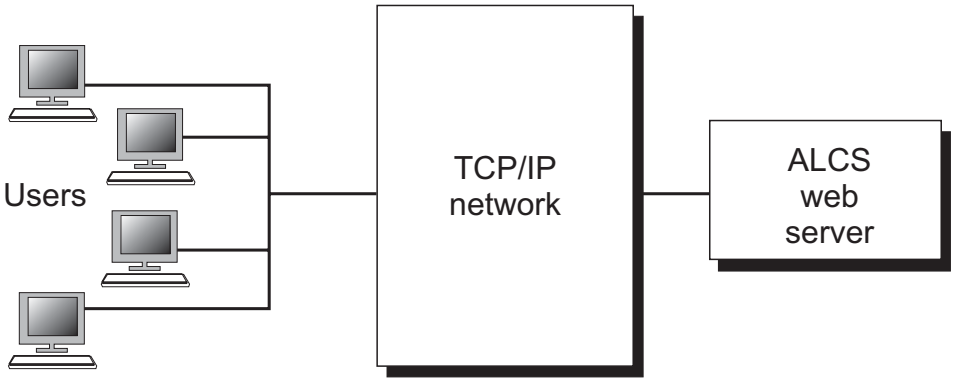


Figure 6. ALCS serving a single network – no firewall

Similarly, if you have no private TCP/IP network, you can use the ALCS Web server to provide information and services to the public Internet with no firewall. (The picture is the same as Figure 6.)

ALCS serving two networks – using a filter

Now consider the case where you have a private TCP/IP network and you want to use the ALCS Web server to provide the same information and services to both the Internet and the private TCP/IP network.

If you connect both TCP/IP networks to ALCS (or, more accurately, to MVS) directly, then you allow free communication (in both directions) between your private network and the Internet.

To prevent this, you can use a network-level firewall (filter) to restrict Internet users so that they can *only* access the ALCS Web server. Figure 7 shows this (to stop the figure getting too complicated, we have left out the Internet users; they connect to the “Public TCP/IP network”).

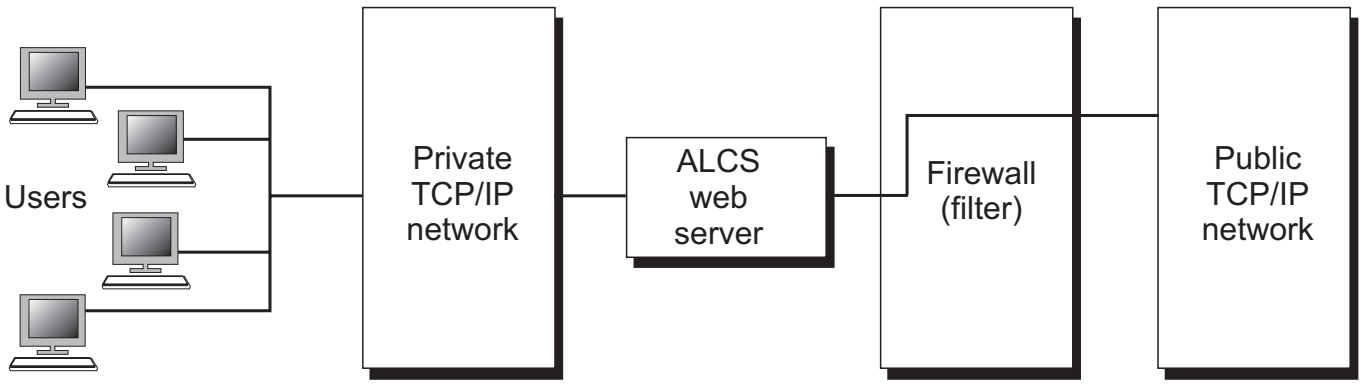


Figure 7. ALCS serving two networks – using a filter

In Figure 7 the firewall allows data to flow *only* between the Internet and the ALCS Web server. This arrangement is safe because the ALCS Web server only provides the information and services that you tell it to provide. In particular, the ALCS Web server does not act as a gateway; you can not send a request to ALCS and have ALCS pass that request on to another server.

Note

It is not necessary to use an external network-level firewall for your MVS system.

Instead, you can run two TCP/IP for MVS address spaces. One connects to the Internet and one to the internal intranet. You can then configure the TCP/IP address space that connects to the Internet so that it acts as a network-level firewall.

Figure 7 on page 23 shows the firewall providing a separate connection joining the ALCS Web server to the public TCP/IP network. This is a *logical* connection, not a physical one. In fact the firewall connects to the private network but it refuses to pass on any data unless the destination is the ALCS Web server (port 80 on MVS's IP address).

The firewall might also do network address translation so that external (Internet) users provide an IP address for the ALCS Web server which is different from the internal (private network) IP address.

Note that the arrangement in Figure 7 on page 23 does *not* allow users on your private network to access Internet services. If you want to allow that, you need a more sophisticated firewall, for example a proxy server or SOCKS server (see 3.4.5, “Proxy servers and SOCKS servers” on page 28).

Servers with different services for different users – using a gateway

Imagine you have several servers (including, of course, an ALCS Web server) and you want to control which users can access the various information and services that your servers provide. Figure 8 on page 25 shows how you can use a gateway to do this.

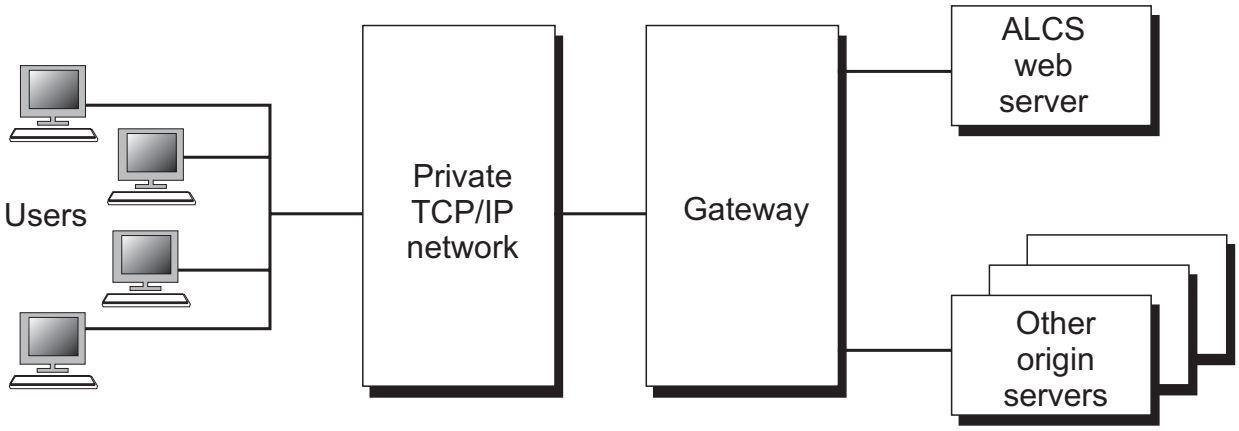


Figure 8. Servers with different services for different users – using a gateway

In Figure 8, the users do not send requests to the origin server that actually provide the information or service they require. Instead they send all requests to the gateway. When the gateway receives a request from a user:

1. It applies your organization's security rules to decide if the user is allowed to request the information or service.

Assuming the user is allowed to make the request:

2. It decides which origin server provides the requested information or service. It might also change the original request.

   For example the gateway might receive a request saying “show me *ALCS's* welcome page”, decide that ALCS should handle the request, and change the request to “show me *the* welcome page”.

3. It sends the request on to the appropriate origin server.

When the origin server responds, it sends its response to the gateway. The gateway sends the response on to the user.

You may be wondering how you prevent users from sending their requests directly to the origin servers. If the users are all your own employees, you may be able to achieve this by relatively simple control of routing tables within your network. But if you want to connect to the Internet (or if you want to restrict your employees' access further) you need a full-function firewall.

Full-function firewall

In “ALCS serving two networks – using a filter” on page 23, we show how you can use a filter to restrict external (Internet) users so that they can access only specific systems in your internal (private) network. You can also use filters to restrict traffic flowing in the other direction. That is, a filter can ensure that only specific systems in your private network can send data to the Internet[1].

[1] We assume that people connected to your internal network do not send data with false (spoof) originator addresses.

In “Servers with different services for different users – using a gateway” on page 24, we show how you can use a gateway to control which users can access the various information and services that your servers provide.

If you want to connect your ALCS Web server (and possibly other TCP/IP servers) to both your private network and to the public internet, and if you want to provide different levels of service to the two networks, you need a firewall that combines the gateway and filter functions. Figure 9 shows this.

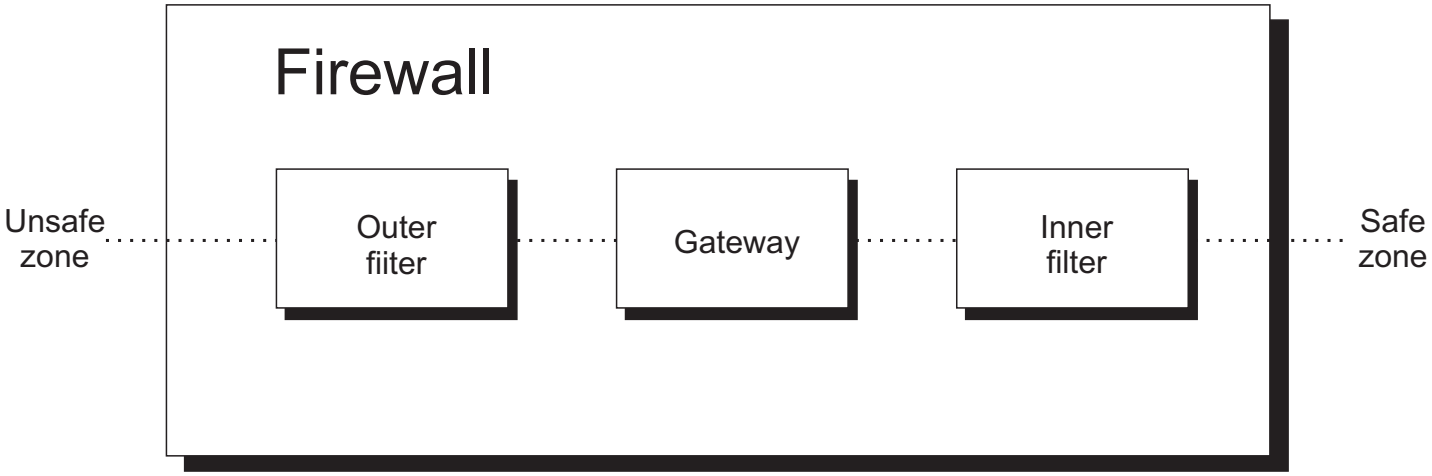


Figure 9. Full-function firewall

In Figure 9, the intention is to allow users in the “unsafe zone” (perhaps the Internet) to access information and services on origin servers in the “safe zone” (perhaps your private network).

The full-function firewall handles requests from the unsafe zone as follows:

1. The outer filter checks the originator and destination addresses of each inbound request.

   The destination address must be a system that inhabitants of the unsafe zone can talk to (this is easy to check – they are only allowed to talk to the gateway). The outer filter can also reject requests from specified originators or, more cautiously, only accept requests from specified originators.

If the outer filter is satisfied that all is well, it passes the inbound request to the gateway.

2. The gateway checks what the request is asking for.

   Assuming that the request is permitted, the gateway changes the request (if required) and sends it on with the destination address of the appropriate origin server.

3. The origin server sends its response back to the gateway.

   You will notice that the designer of this firewall is not too sure that the “safe zone” really is safe. They have included an inner filter that can reject incorrectly addressed responses.

4. The gateway sends the reponse on to the user agent (the original requestor).

3.4.3 Firewall questions and answers

By now (if you are still reading this!) you may be asking yourself questions such as:

- **Is that all, or are there yet more complex firewall configurations?**

  No. That is not all. There are much more complex firewall configurations. For example, some firewalls include an outer and an inner gateway.

  There are also completely different types of firewall (we discuss two of them in 3.4.5, “Proxy servers and SOCKS servers” on page 28).

- **Do I need a full-function firewall (or something even more complex)?**

  Possibly. It depends on what connects to your “safe zone” (what you are trying to protect) and what you consider to be the threats from the “unsafe zone”.

  You will probably hear widely differing opinions on this subject. Even security specialists do not agree with each other. For example, some experts argue that:

  > The Internet is an extremely dangerous place, populated by large numbers of “hackers” (dedicated to destroying your organization for fun) and criminals (dedicated to abusing your systems for personal gain). If you connect to the internet at all, you need the most sophisticated security systems that money can buy.

  while others argue that:

  > The Internet is not much more dangerous than any other way of doing business. Very few “hackers” intend to do any damage and most criminals prefer simpler ways of committing their crimes. What you invest in security should be governed by common sense rather than paranoia.

  If you are unsure about this (or other security questions) you should talk to your IBM representative.

- **How much will all this cost?**

  Prices for different firewall configurations vary widely. We suggest you discuss this with your IBM representative.

- **If I install a full-function firewall, is my system 100% safe?**

  No. Firewalls only protect against certain types of “attack”.

  For example, a firewall does not protect your organization against conventional burglary, blackmail of your employees, and so on. Similarly, if you put confidential information into one of your Web pages, the firewall does not automatically prevent access to that page (you must tell the firewall which pages to protect).

3.4.4 Using a firewall to isolate parts of your private network

All the examples in 3.4.2, “Example firewall configurations” on page 22 assume you want to use a firewall to separate your private network from the public Internet.

You can also use a firewall to isolate one part of your own private TCP/IP network from another part.

For example, you might split your network into a “high-security” network and a “medium-security” network, using a firewall to separate the two. You can then

attach any servers that hold highly confidential information to your high-security network and use the firewall to prevent anyone attached to your low-security network from accessing them.

3.4.5 Proxy servers and SOCKS servers

Proxy servers and SOCKS servers are specialized firewalls that allow staff *inside* your organization to use TCP/IP servers *outside* your organization.

These servers do not normally allow *any* access in the reverse direction.

That is, if you connect your private intranet to the public Internet with a proxy or SOCKS server, then users on your private intranet can access servers on the Internet. Users on the Internet can not access servers on your intranet.

If you want to allow Internet users to access selected servers on your intranet, you use a conventional firewall. If you want to allow your intranet users to access servers on the Internet, you use a proxy or SOCKS server. If you want to allow both types of access, you need both types of firewall.

3.5 Financial transactions on the Internet

If you plan to have your Web server *sell* goods or services, you need to consider how your customers will pay. There are at least two obvious ways you can arrange this:

- Your customers send their order for goods or services to your Web server. Then they pay by mail or telephone – possibly using a credit card or charge card – or in person, at an appropriate agency.
- Your customers send their order, together with some form of payment, to your Web server. The payment might consist of credit or charge card details.

If you only use your Web server to sell to users who connect through your private intranet, either of these methods may be acceptable. But you may want to sell to users who connect through the public Internet. The Internet does not provide a secure connection between your customer and your Web server. Unless you take special precautions, this can have serious implications, including:

- Your customers may be unwilling to transmit credit or charge card details over the Internet. They may be worried that “hackers” could get the information and abuse it.

This is an obvious and widely understood exposure, but you also need to consider that it is possible to “fake” an Internet address. This means that:

- Your Web server can not assume that the originator of the request is genuinely the customer they appear to be.
- Your customer can not assume that the Web server they are communicating with is a genuine provider of goods or services.

To conduct financial transactions across the Internet, customers need to be sure that the vendor is who they appear to be and vendors need to be sure that customer is who they appear to be.

If you chose to have your customers just place orders on the Internet and then pay by telephone, by mail, or in person, then you do not need to worry too much about this problem. If a customer does not pay then you can simply cancel the order.

If you want to allow electronic payment over the Internet, then you (and your customers) must either accept a degree of risk, or you must use specialized encryption and authentication technology. A full discussion of this technology is beyond the scope of this document, but the following paragraphs give an idea how it works. If you want to know more about securing financial transactions on the Internet, IBM can provide you with specialized advice.

3.5.1 Public-key cryptography

A technique called **public-key cryptography** uses two keys; a public key (for encrypting messages), and a private key (for decrypting messages). It is almost impossible to use the encryption key to find out the corresponding decryption key.

Public-key cryptography allows your enterprise to tell *everyone* how to encrypt a message that only your enterprise can decrypt. All you need to do is tell them the public key.

Your customers can now send messages (including credit card authorization) and be sure that no-one except you can understand it.

Similarly, a customer can give you his or her public key so that any reply you send is meaningless to anyone other than your customer.

Of course, a bogus customer could send you an order with details of someone else's credit card. They ignore your response (which they do not “understand” because they do not know your real customer's private key). To avoid this problem, you need some form of “electronic signature” which proves the originator of a message is who they say they are.

3.5.2 Electronic signatures

Public-key cryptography allows you to transmit an electronic signature. It works something like this. If you encrypt a message with *your private key* then the receiver can decrypt the message with your public key. The receiver can be sure the message comes from you (or at least from someone who knows your private key). Of course, *anyone* who knows your public key can decrypt your message. But you can encrypt your message with your private key *and* with the receiver's public key. Now only your intended recipient (or someone else who knows their private key) can decrypt the message.

This provides a secure electronic signature, provided that:

- Your customer knows for sure that “your” public key really belongs to you (only you know the private key).
- You know that your customer's public key really belongs to your customer (only your customer knows the private key).

To allow you (and your customers) to be sure, a number of agencies provide key-registration services. Anyone can register a key – or more usually, get a key – from one of these agencies. The agency then warrants that only the authorized

user knows the private key; that is, if you encrypt using a particular public key, only the registered owner can decrypt.

3.5.3 Data Encryption Standard

Public-key cryptography has a number of features that make it especially useful for securely transmitting information on the Internet. In particular:

- It does not need a secure way to send encryption keys – the encryption key is public.
- It allows electronic signatures.

Unfortunately, public-key cryptography is expensive to use for large messages. The encryption and decryption process uses a lot of computer power or time or both.

It is much cheaper to use the Data Encrytion Standard (DES) for transmitting bulk data. DES uses the same key for encryption and decryption, consequently, it requires a secure way to send the key between the sender and recipient.

The usual way to achieve good security at reasonable cost is to use both public-key and DES cryptography. You use public-key cryptography where necessary, for example to send DES keys and electronic signatures, and you use DES cryptography for the rest.

3.5.4 The HTTP Server for MVS

The ALCS Web server does not provide cryptography services. If you want to use cryptography to protect communication between the ALCS Web server and client browsers, you can use the IBM HTTP Server for MVS.

The HTTP Server for MVS provides a variety of sophisticated security functions and can take advantage of the special cryptography hardware available on IBM's System/390 processors. Figure 10 shows how you use the HTTP Server in combination with the ALCS Web server.

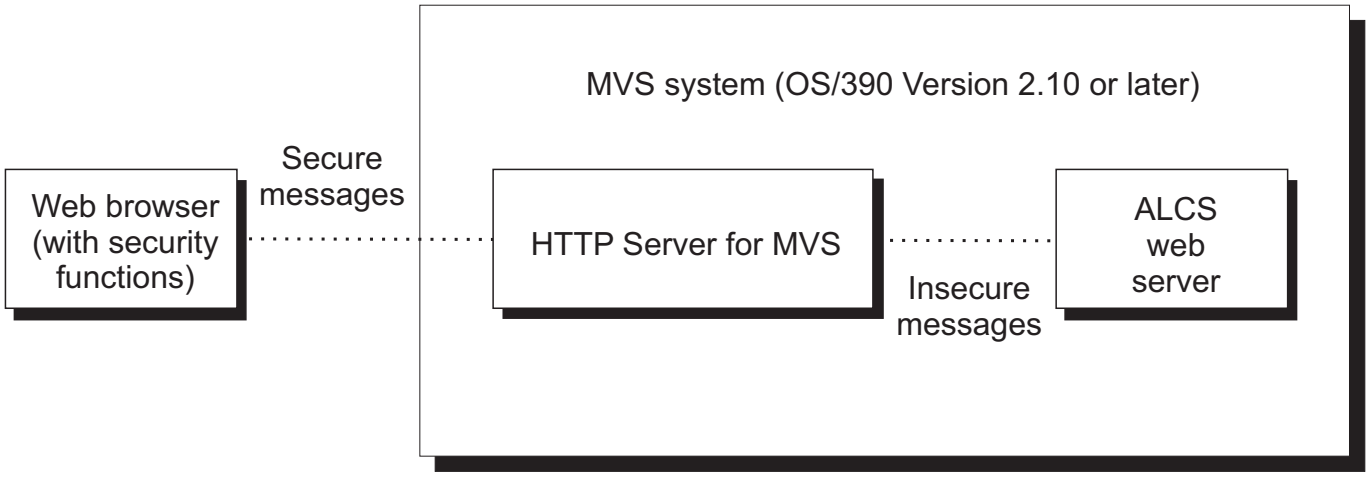


Figure 10. Using the IBM HTTP Server for MVS with the ALCS Web server

Of course this only works with client Web browsers that implement corresponding security functions, including the appropriate encryption and decryption algorithms. Many popular Web browsers do have these functions, but people with other browsers can not send or recieve secure messages. For more information about this type of configuration, contact IBM.

Chapter 4. Designing an ALCS Web site

Designing a Web site, whether or not you plan to use ALCS as your Web server, requires some planning. You probably want to make your Web site easy to use and interesting (Internet users call this “cool”). This is especially likely if you plan to allow public access to your site through the Internet.

There are a number of specialist organizations that can help you with designing your Web site. But you can learn a lot about how to achieve these apparently simple objectives by exploring other Web sites – especially sites that specialize in subjects very different from what you plan.

You will notice very quickly that:

- The welcome page is important.

  There are a vast number of sites, and if a welcome page does not interest you, you are likely to move on to another site. A welcome page should show the information and services available, and provide hyperlinks that get you quickly to the pages you want to see.

- Images and multimedia effects can make a site interesting.

  It is astonishing how much difference even a simple background image makes.

- Images and multimedia effects can take *ages* to receive.

  If you have to wait a long time for a page to arrive, this can put you off visiting the page again. It can be a mistake to include too much of this stuff in your welcome page.

  Experienced Web users often tell their browser not to load graphics (and multimedia effects) while they explore. They only load the graphics when they find a page that looks interesting. This means your page must “work” without graphics.

- Some Web pages may not work with your Web browser.

  Some sites offer you a choice of pages so that you can select ones that best suit your browser. Others require you to install a specific Web browser. Ask yourself the question:

  > Am I happy to get and install a new Web browser so that I can use this Web site?

Tip

If you see a way of presenting information that you like, you can use your Web browser to look at the HTML for the page. You can then copy the technique in your own Web pages.

4.1 Overall design considerations

When you visit a Web site, you may not know much about the organization that owns the site, and you may know even less about what information and services the site provides.

- You want to design your site so that it is a good advertisement for your organization and for your Web site.

When you visit a number of Web sites (or even one Web site) you can sometimes get “lost”. There are often several hyperlinks on each page, including some that refer you to pages on different Web sites. You may not be too sure which hyperlinks lead you to the information or service you want. If that happens, you can quickly end up not knowing what page you are looking at or how to get back to (or forward to) the page you want.

- You want to design your site so that it is easy to get to the page a particular user might want, and easy for your users to know what each page is.

4.1.1 Site style

Your organization may have a “house style” for publications. A house style often includes a standard representation of your organization's logo, and preferred paper colors, fonts, layout styles, and so on. If you do have a house style, you may want to use it for your Web pages.

Even if your organization does not have a house style, it is generally a good idea to use a similar style for all the Web pages on your site. For example, it is generally a good idea to use the same:

company logo
colors (background and text)
background images (“wallpaper”)
dingbats[2]
and so on

for all your pages.

A consistent style helps visitors know that a particular page is part of your site (they may wonder if a hyperlink has taken them to a different site).

Also, if you use the same graphic more than once (for example, if you use the same background image for all your pages) most browsers request the graphic only once. This makes it *much* quicker for your users to move from one page to another.

In addition to using a consistent style for all your pages, you may want to include some “standard” information in every page. This might include:

- The name of your organization
- A hyperlink “back” to your welcome page

[2] A dingbat is a small image or icon that you use to decorate your pages. For example, you might use a telephone dingbat in a list of telephone numbers, like this:

☎ 0123-456 7890
☎ 0123-456 7891

- Other “standard” hyperlinks
- A copyright statement.

| You can conveniently do this with server side includes (see 8.4, “Server Side
| Include (SSI)” on page 97).

Finally, do not forget that you can not force a Web browser to present your pages exactly the way you intend. For example:

- A browser can not show your carefully chosen colors if the user has a monochrome screen.
- A visitor may tell their Web browser not to load images. You should include text alternatives for images (especially if the image conveys important information, like the name of your organization).

We have more to say about this in 4.2, “Browser dependencies” on page 41.

4.1.2 Welcome page

Most (but not necessarily all) visitors to your site will start with your welcome page. If they do not like it, or do not understand it, they may decide not to explore your site and they may not visit your site again.

The following suggestions are intended to help you avoid this problem, but they are only suggestions, not rules. You may have your own reasons for doing things differently.

Say who you are: Be sure to include the name of your organization at or near the top of your welcome page. If you use your logo for this, remember to include alternative text in case your visitor is not loading graphics.

It may be wise to include a brief description of what your organization does, or a hyperlink to a page that describes your organization, or both.

Say what your site does: The usual way to do this is to include a set of hyperlinks that lead to your information and service pages. The wording of these hyperlinks is important – be sure to make clear what is on your site and what is not.

A visitor who does not immediately see a link that looks promising may not be motivated to explore.

A visitor who spends a long time searching your site for something that is not there may become frustrated and not visit your site again.

Keep it small: If your page takes too long to arrive, your visitors may lose patience. Most browsers allow the user to stop loading a page if they get bored – and even if they do wait for your page, a very long load time might put them off visiting again.

Be careful with graphics: Graphics can make your home page much more interesting than plain text, but remember that large complex graphics objects take time to load. And remember that some visitors do not load graphics (at least, not until they decide your site is interesting).

Keep it simple: You may want to make your welcome page more interesting (cool) by using the special capabilities of a particular Web browser. But remember that a visitor who does not have that Web browser may not be able to understand your welcome page. It is not unusual for someone with a low-function browser to visit a welcome page that displays a completely blank screen (this is not particularly cool).

4.1.3 Site structure

If your site includes more than a very small number of pages, you need to think about how you organize information and services into pages, and how you organize these pages in a way that makes it easy for your visitors to find the pages they want. To do this, you need to think about:

- What information and services your site provides.
- What visitors you want or expect.

Of course these are not independent. The information and services your site provides affects who visits your site. And once you have decided who you want to visit your site, that affects what information and services you provide.

Also, your site will probably change over time. You may want to make changes when you see how visitors use your site (to make it easier to use), you may want to make changes to encourage people to visit your site regularly (to see what is new), and so on.

For example, if you are a sporting goods manufacturer, you might want to include the following on your site:

- Information about your organization.
- The history of your organization.
- Famous personalities who use your products.
- Famous past events where your products were used.
- Future events where your products will be used.
- Testimonials from satisfied customers.
- Photographs and technical details of each of your products.
- A list of retailers who stock your products.
- News about recent and planned additions to your range of products.
- Descriptions of the sports (what exactly *is* cricket?).
- Reviews of your products from newspapers and magazines.
- A current price list.
- Information about special offers.
- Pages where your visitors can order your products.
- Pages where your visitors can request more information about your products.
- And so on.

Each of the above might include images (photographs, drawings, and so on), audio and video clips, and other multimedia objects. It is easy to see that your site contains a lot of information.

A popular way to organize infomation is to use a hierarchy. For example:

- About our company...
  - Acme Sporting Goods Inc. today
  - The history of Acme Sporting Goods Inc.
  - What our customers say about Acme Sporting Goods Inc.

  - What the media say about Acme Sporting Goods Inc.
  - and so on
- About our products...
  - Exciting new products
  - Special offers
  - A complete list of our products
    - Athletics
    - Ball games
    - Gymnasium
    - Keep fit
    - Swimming and water sports
    - and so on
  - Special services
    - Engraving
    - Presentation packs
    - and so on
- Everything you want to know about sport...
  - The rules...
    - Cricket
    - Fives
    - Synchronized swimming
    - and so on
  - The benefits of physical fitness
  - Getting the most out of sport
  - and so on
- How to buy our products...
  - A list of retailers who stock our products
    - Africa
    - Asia
    - Australasia
    - Europe
    - North America
    - South America
  - Buying direct from us
    - Ordering by mail or telephone
    - Order now from your Web browser
    - Terms and conditions
    - and so on

(You might have odd bits and pieces of information that you want to provide which do not “fit” into the hierarchy. We suggest a way to deal with these in 4.1.5, “Frequently asked questions” on page 41.)

Now consider who might visit your site. We can imagine many types of visitors, including:

- A professional cricketer who wants detailed technical information about a particular cricket bat.

  You want to provide this specialized information quickly and efficiently so that the cricketer decides to buy your product.

- A sports club representative who is looking for a “one stop shop” for a wide range of sporting goods.

  You want to show the range, quality, and attractive pricing of your products so that the club selects you.

- A parent who is looking for a birthday present for their child.

  You want to convince them that you have products which are ideal presents – somthing that will delight their child and will not cost too much.

- A school who wants a specially engraved javeline for a prize.

  You want to tell them you can provide this (without them having to spend hours searching your site).

- Someone writing a book or magazine article about sports, who is looking for information that they can use.

  You want them to mention your name as a famous company with a long history of supplying high quality goods to famous sporting personalities.

- Someone who stumbled across your site while looking for something completely different[3].

  You want to impress them so that when they *do* want something you provide they know where to come.

- And so on.

As we mentioned in 4.1.2, “Welcome page” on page 35, your home page normally includes a set of hyperlinks that lead to your information and service pages. In our sporting goods example, your home page might include the following hyperlinks:

- **About our company...**
- **About our products...**
- **Everything you want to know about sport...**
- **How to buy our products...**

If you follow the first hyperlink, “About our company...” then you get to another list of hyperlinks, like this:

About our company...

- **Acme Sporting Goods Inc. today**
- **The history of Acme Sporting Goods Inc.**
- **What our customers say about Acme Sporting Goods Inc.**
- **What the media say about Acme Sporting Goods Inc.**
- **Return to Acme Sporting Goods' home page**

We have arranged the pages of our site into a hierarchy, with a set of nested menus. This structure has the advantages of simplicity and familiarity (it is similar

[3] You might be visited by a biology student searching for information about bats (the flying mammals).

to the way you access the information in a book from the table of contents). Unfortunately it also has disadvantages, including:

- It forces our visitors to select from a succession of menu pages that are not particularly interesting.

  A visitor might find this boring and frustrating ("Will I *ever* get to the actual information?").

- It forces our visitors to explore our site in a fixed order.

  A visitor might see a favorable quote about our cricket bats in "What the media say" and want to jump straight to a list of the bats we supply (and from there, straight to our on-line purchasing page, we hope!).

  We help a little bit by allowing our visitors to jump straight back to our home page from wherever they happen to be.

There are several things we can do to improve on our simple hierarchical page organization. These include:

- When we organize our information and services we use a hierarchical model. For example, we think of the way we access information in a book (using the table of contents), or the way we access TSO facilities (using menus).

  This is a good way to *organize* our information, but it is not necessarily a good way to *present* our information.

  In our example, "About our company..." displays a simple list of hyperlinks. Instead of this simple list, we can display a page that gives an overview of Acme Sporting Goods Inc. and includes graphics (possibly a photograph of our factory).

  This page can still act as table of contents. We do this by including hyperlinks to the pages that provide more detail within the text or graphics (or both).

- Even though we want a page to act as a table of contents, we can include hyperlinks that do not follow our hierarchical structure.

We might make our "About our company..." page look something more like this:

About our company...

Acme Sporting Goods Inc. is one of the worlds leading suppliers of sporting goods. We have a **long tradition** of providing first class **equipment** and **services** to some of the worlds **leading sports personalities**. But we do not rest on our laurels. **Acme Sporting Goods today** is a thoroughly modern company.

You do not have to take our word for it. You can read what leading **newspapers and magazines** and **customers** say about us.

- **Return to Acme Sporting Goods' home page**

Text that displays like **this** is a hyperlink. For example, **long tradition** links to our "The history of Acme Sporting Goods Inc." page. You will notice that we have included all the "table of contents" hyperlinks that we had previously. We have also added some links that do not follow the simple hierarchy. For example, **equipment** links directly to information about our products.

4.1.4 Feed-back

In addition to browsing information, reproducing multimedia effects, and sending forms data, many Web browsers provide an electronic mail facility that allows the user to send a message.

A common way to exploit this is to include a hyperlink in your home page (or in every page, if you wish) that requests comments about your Web site. For example, you might include:

```
<p>Please let us know what you think of our site.
Send comments to:
<a href="mailto:comments@acme.com">comments@acme.com</a>
```

which displays something like:

Please let us know what you think of our site. Send comments to: **comments@acme.com**

If their Web browser provides an electronic mail facility, the user can simply click on **comments@acme.com** to send their comments. This displays a standard input form that the Web browser provides; you do not need to maintain the form on your Web site. If their browser does not provide an electronic mail facility, or if the user prefers some other way of sending electronic mail, they can make a note of the address (comments@acme.com).

You can also exploit this facility to allow users to send you questions that they have about your site or your organization. Something like:

```
<p>If you have any questions about this site or about Acme Sporting Goods,
please check out our <a href="/faq/faq.html">frequently asked questions (FAQ)</a> page.
If you do not find the answer there, please send your question to:
<a href="mailto:questions@acme.com">questions@acme.com</a>.
```

which displays something like:

If you have any questions about this site or about Acme Sporting Goods, please check out our **frequently asked questions (FAQ)** page. If you do not find the answer there, please send your question to: **questions@acme.com**.

You will notice that we have included a hyperlink to something called our “frequently asked questions (FAQ) page”. We describe this next.

4.1.5 Frequently asked questions

Many Web sites include a page, or sometimes several pages, of miscellaneous information arranged as a set of questions and answers. The usual name for this is frequently asked questions (FAQ).

If you provide an electronic mail address for questions (see 4.1.4, “Feed-back” on page 40), you can add the questions that you receive (and the answers, of course) into your FAQ.

You can also use your FAQ for odd bits and pieces of information that you want to provide, but which do not “fit” into the hierarchical organization of your site; of course, you will have to make up the questions yourself.

4.2 Browser dependencies

If you intend to allow access to your Web site from the Internet, you need to be aware that there is a bewildering variety of Web browsers (over 100 reasonably popular ones last time we checked, and the number is increasing). Different browsers have different capabilities that can affect the way they present your Web pages. For example:

- Some browsers can not display graphics.
- Some browsers can not display HTML tables.
- Some browsers can only display US English characters.

Not only are there many different browsers, but some browsers support a variety of viewers or plug-ins that extend the browsers capabilities. For example a browser might support special viewers for:

- Multimedia effects.
- PostScript documents.
- Languages other than US English.

A user with a particular browser may or may not have these browser extensions.

Even the same browser can behave differently in different situations. For example:

- A user can tell their browser not to load graphics.
- A user may have a monochrome display.

Finally, different users may have different preferences or requirements for the way you present your information. For example:

- Different users may prefer or require information in different languages.
- Some users may prefer or require information in Braille or speech.

Of course, you may need to think about these different capabilities and requirements even if you only intend to allow access to your Web site from your private intranet.

There are several ways to deal with this problem. We discuss some of them in the following paragraphs.

Design for the simplest browsers: You can design your site so that all the pages can be used on the simplest browsers. The simplest browsers only support the US English alphabet, can not reproduce images or multimedia effects, and only support the simplest HTML.

If you chose to do this, you can still include graphics and multimedia effects in your pages; simple browsers ignore them. But you do need to be sure that your pages "make sense" without them.

For Anyone who visits your site can view your Web pages.

Against Your site may not be very interesting (cool).

You can only use a limited number of languages (the ones that use the US English alphabet).

Design for specific browsers: You can select a specific browser (or possibly more than one) which provide facilities that you want to exploit. You can then design your site so that it works with your selected browser or browsers.

If you chose to do this, you can include hyperlinks on your home page that direct users to a site where they can download or purchase your selected browsers.

Of course, if your page does not display anything at all on some browsers, those users will not see the hyperlink. You can avoid this problem by having a dynamic welcome page. The Web program that processes the welcome page request checks the originating browser and generates a special page if the request does not come from one of your selected browsers. The special welcome page can contain the hyperlink to your selected browser site.

For You can use the latest fancy HTML features, multimedia effects, and so on to make your site interesting.

Against Some visitors may be unable or unwilling to install a particular Web browser. Those visitors will not use your site.

Customize for specific browsers (automatic): You can design your site so that it has different sets of pages, each customized for a particular browser or set of browsers. For example, you could have one set of pages that exploit HTML tables, frames, and so on and rely heavily on multimedia effects, and another set of pages that work on simple browsers.

You can then have a dynamic welcome page. The Web program that processes the welcome page request checks the originating browser and selects, or generates a custom welcome page for that browser.

For You can selectively use HTML features, multimedia effects, and so on to make your site as interesting as possible for all your visitors.

Against You must write a Web program to select or generate your welcome pages. Because there are so many browsers, and browsers are regularly enhanced, it is difficult to be sure your Web program classifies browser capabilities correctly.

Customize for specific browsers (by menu): You can design your site so that it has different sets of pages, each customized for a particular browser (or set of browsers).

You can then have a welcome page that works on the simplest browsers but includes selection of hyperlinks that lead to your custom pages. This allows the user to select pages that work with their browser and avoids you having to check the originating browser in your Web programs.

For You can selectively use HTML features, multimedia effects, and so on to make your site as interesting as possible for all your visitors.

Against You may have to include a long, and possibly confusing, list of options in your welcome page.

Other customization and menu options: Whichever way you chose to deal with the problem of differing browser capabilities, you may want to include additional options on your welcome page (or pages, if you have custom welcome pages). For example, you may want to include hyperlinks to a text-only version of your site, to different language versions of your site, and to versions of your site that cater for users with special needs (for example, users with impaired sight or hearing).

Tip

If possible, try to view your Web pages with more than one Web browser. You can down-load several Web browsers free from the Internet. Also try to view them both with and without images and other multimedia effects.

4.3 Designing Web transactions

For most of this chapter, we have been discussing relatively simple Web server functions. That is, the Web server receives a request from a browser and returns a Web page. These can be either static pages (the URL specifies a file in the ALCS hierachical file system) or dynamic pages (the URL specifies a program in the ALCS hierachical file system, the program selects, or builds the Web page).

A special characteristic of the ALCS Web server is that you can use it to interface to your existing ALCS applications.

Most ALCS applications are designed to handle transactions. That is, an end user sends input messages, the ALCS application programs process these input messages and generate responses.

4.3.1 Simple transactions

The simplest type of transaction is where the end user requests some specific information, the application programs extract that information from the data base, build a response message that contains the data, and send the response to the end user. A typical example is the schedule display transaction implemented by airline reservation systems. We use the IPARS schedule display transaction as our example in the following discussion.

Input for simple transactions

The input message that requests the IPARS schedule display includes information about *which* schedule to display. For example, the input message:

```
S 15SEP LON ZRH 1200
```

requests the schedule for flights from London (LON) to Zurich (ZRH) on the 15th September at or around 12:00 hours (local time in London).

If we want to implement the IPARS schedule display as a Web transaction, we need some way for the user to specify this information. We could simply present a form with an input field and ask the user to enter the message; something like this:

```
<p>Please enter your schedule request here:
<input type=text name=input size=20 maxlength=20>
```

When the user submits this forms data, we simply pass that to the IPARS input message editor.

There are a number of obvious disadvantages to this method, including:

- It assumes the user knows the format of IPARS input messages. It is highly unlikely that anyone would know this, unless they are trained to use IPARS.
- It assumes the user knows the standard airline abbreviations for cities and airports. Again, it is unlikely that many users would know these.
- Unless we take special precautions, it allows the user to enter *any* IPARS input message. There are probably many input messages we would prefer Web users not to enter (for example, messages that retrieve information about other passengers).
- It assumes the user already knows where they want to go. This may be true for some users, for example people planning a business trip. Others users, for example people thinking of taking a weekend break, might prefer to see a list of options.

We can solve many of these problems by providing a form that includes selection lists; something like this:

```
<p>Please select the date you want to travel and the cities you want to travel between.
<p>Date:
<select name="day" size=1>
<option> 1<option> 2<option> 3<option> 4<option> 5<option> 6<option> 7<option> 8
<option> 9<option>10<option>11<option>12<option>13<option>14<option>15<option>16
<option>17<option>18<option>19<option>20<option>21<option>22<option>23<option>24
<option>25<option>26<option>27<option>28<option>29<option>30<option>31
</select>
<select name="month" size=1>
<option value="JAN">January <option value="FEB">February <option value="MAR">March
<option value="APR">April   <option value="MAY">May      <option value="JUN">June
<option value="JUL">July    <option value="AUG">August   <option value="SEP">September
<option value="OCT">October <option value="NOV">November <option value="DEC">December
</select>
<p>From:
<select name="board" size=3>
<option value="ANK">Ankara
<option value="CAN">Canakkale
<option value="IST" selected>Istanbul
<option value="IZM">Izmir
</select>
To:
<select name="off" size=3>
<option value="ANK" selected>Ankara
<option value="CAN">Canakkale
<option value="IST">Istanbul
<option value="IZM">Izmir
</select>
```

Note that with our new improved input form:

- The user does not need to know the format of IPARS input messages. They only select the information that the IPARS application requires.
- The user does not need to know the standard airline abbreviations for cities and airports. They select from a list of ordinary names like “Canakkale” that they do know (especially if they live in Turkey!).
- We do not need special precautions to prevent users from entering “undesirable” input messages. They do not enter IPARS messages at all.
- The user does not need to know where they want to go. They can check our list of the places we fly to.

Our new improved input form still only uses very basic HTML. We could make it even easier to use (and more fun) by, for example, using Java.

Output for simple transactions

Now that we have received the input we need, we pass it to the IPARS schedule display application. The application creates a response that shows the selected schedule. We need to send this information back to the Web browser.

We could simply send the response that IPARS generates. But this requires the user to understand the specialized abbreviations (three-character aircraft types, one-character class codes, and so on) that IPARS uses. We can make the

response more useful and much easier to understand by decoding the abbreviations.

And we could add hyperlinks to images (or movies) of the aircraft, the destination cities, and so on. Some browsers support “virtual reality” models; we could have the ALCS Web server send a model that allows the user to explore the destination airport, the interior of the aircraft, or whatever we chose.

4.3.2 More complex transactions

Transactions like the IPARS schedule display are simple to implement because each transaction is just one input message and its response. But many ALCS applications support transactions where the user enters a series of input messages which relate to each other. For example, a seat reservation transaction proceeds something like this:

```
    Input:  Display the availability on flights from London to Zurich
            on 15th September at around 12:00 hours
Response:   Shows a selection of flights, numbered 1, 2, 3, and so on
    Input:  Book 3 seats in first class on the 2nd flight in the list
Response:   OK
    Input:  The three passengers are all called Hobson
Response:   OK
    Input:  The passengers' telephone number is 818 4203
Response:   OK
```

and so on.

The actual messages are more difficult to understand than what we show, and there are lots of complicated things we have left out of our dialog. But it is clear that putting this transaction on the Web is more complex than our simple schedule display.

In particular, all the information we need to generate a schedule display arrives in a single input message. For this booking, we need to collect information from a whole series of input messages. We could ask the user to submit all the information for the booking in a single form and write a program that translates the form input into the corresponding sequence of IPARS input messages. In fact this is probably the best way for the user to give us details about them (their name or names, telephone number, address, special meal requirements, and so on).

It is probably *not* the best way for the user to give us details about the flights. Most likely the user wants to see a selection of flights from A to B and select one, then see a selection of flights from B to C and select one, an so on. It is much more convenient for the user if we remember these selections inside ALCS. If we do not then the user must make notes while they select their flights and enter all the details (flight numbers, with corresponding departure points and arrival points, dates and times, and so on) in the booking input form.

Another possible problem, which airlines know well, is that a user might decide on a particular flight (from A to B) which has seats available. Then they decide on another flight (from B to C). Eventually they complete their trip plan and fill in the form with all the details. Unfortunately there is the chance that by this time we have sold all the seats on the flight they selected for A to B.

This is much less likely if we allow the user to book the seats on each flight at the time they select it. If they change their mind (always possible) then we can "unbook" the seats. Existing airline reservation applications, including IPARS, are designed to work like that.

It is obvious that we need to think carefully about exactly how this Web booking transaction might work. Possibly you have a clear picture of how we can do this, or you have some ideas but are a bit vague about the details, or you are thinking the whole thing is far too complicated and you already have a headache. In any case, we now outline one possible plan.

4.3.3 Example complex transaction – reserving airline seats

We start off assuming that somewhere (possibly several places) on your Web site you have a hyperlink something like:

> Time flies, but so do we – why not **book your trip now**

This hyperlink takes the user to an introductory page.

Reserving airline seats – introductory page

Our introductory page includes lots of encouraging stuff and some hyperlinks. Something like this:

> Thank you for chosing to book your trip with Acme Sporting Goods Inc. We are sure you will enjoy our outstanding service, comfortable aircraft, and in flight sports movies. You may want to **browse our schedules** or check our **special offers** first.
>
> If you are not already a member of our *Javelin Club*, why not check out **benefits of Javelin Club membership**.
>
> If you are new to this service, you might want to read **how our reservations on the Web works** before you start.
>
> If you want to book a trip for more than three people, please **contact us directly**.
>
> If you are ready to start now, please complete the following details. You can book a trip for up to three people. If you are travelling alone, just provide your own name. Otherwise, provide your own name and the names of the one or two people who will be travelling with you.
>
> (Input fields appear here.)

The hyperlinks in this page link to other pages as follows:

browse our schedules
: Links to a schedule display form. We already described how this might work in 4.3.1, "Simple transactions" on page 43.

special offers
: Links to a page that describes our special offers.

benefits of Javelin Club membership
Links to a page that describes the wonderful benefits of our Javelin Club for frequent fliers.

how our reservations on the Web works
Links to a page that describes how our reservations on the Web works.

contact us directly
Links to a page that provides telephone numbers, mailing addresses, and so on.

Following the text on this page, we include a form where the customer enters details such as:

- Passenger names, with title (Mr, Ms, Dr, and so on) and age-group (adult, child, or baby)
- Postal address and telephone number
- Special meal requirements (vegetarian, gluten-free, and so on)
- Special assistance requirements (for example, wheel-chair)

and so on.

Assuming our customer decides to proceed, they complete these details and press a submit button. This sends the details to our application (in ALCS) which saves them on the data base.

Reserving airline seats – confirming customer details

After our application saves the customer's details, we want to reassure them that we have the correct information and offer a chance to correct any errors.

To do this, our application sends a response that shows the information we have stored and offers three options:

- Ooops, I want to correct my details
- Forget it, I want to cancel my reservation
- OK, I want to proceed

"Ooops, I want to correct my details" presents a form similar to "Reserving airline seats – introductory page" on page 47. The form shows the details we have; the customer can correct them as required and submit again.

"Forget it, I want to cancel my reservation" discards the information our application has saved and presents a simple text page which tells the customer that we have cancelled the transaction and discarded their details.

"OK, I want to proceed" presents the first of two forms which allow the customer to select a flight.

Reserving airline seats – selecting a flight

We use two forms to allow the customer to select a flight.

On the first form, the customer selects where they want to fly from (the board point), where they want to fly to (the off point), and the date (and possibly time) they want to fly. This form is much like the one we use for schedule selection in 4.3.1, "Simple transactions" on page 43 – except we probably want to include some of the information that we already have about this trip. This helps the

customer keep track of where they are. For example, we might include an introductory paragraph that includes:

- The customer's name, and how many people we are booking seats for.
- The flights we have already booked for this trip. (The first time we show this form, there are none. But we use this same form again for the second and subsequent flights of this trip.)

The customer selects the board and off points and the date, and submits the information. (We might also want to include an extra submit button in case the customer wants to change one of the flights they already booked.)

Our response is a list of suitable flights and includes our second form. The customer selects the flight they want and submits. We probably want to provide more than one submit button, to allow:

1. Book this flight and let me chose the next flight in my trip
2. Book this flight and that is the end of my trip
3. I changed my mind, let me select another destination (or date, or whatever)
4. Forget it, I want to cancel my whole reservation

For option 1, we reserve the seats on the selected flight and display an updated version of the first of our “selecting a flight” forms.

For option 3, we redisplay the first of our “selecting a flight” forms.

For option 4, we release any seats we already reserved for this trip and present a simple text page which tells the customer that we have cancelled the transaction.

For option 2, we display our finalization form.

Reserving airline seats – finalization

Eventually (we hope) our customer books seats on all the flights they require. On their last flight selection, they enter “Book this flight and that is the end of my trip”.

We present a finalization page (form) that shows all the details we have about the trip and offers the following options:

- OK – finalize the booking
- Ooops, I want to correct my details
- Ooops, I want to change one of my flights
- Forget it, I want to cancel my whole reservation

Reserving airline seats – summary

Figure 11 on page 50 summarizes the pages and forms that we use to implement our airline seat reservation transaction on the Web. It does not show all the details that we included earlier. By now, you probably have your own ideas about how you can extend or improve our implementation – and we do not want to make the picture too complicated.

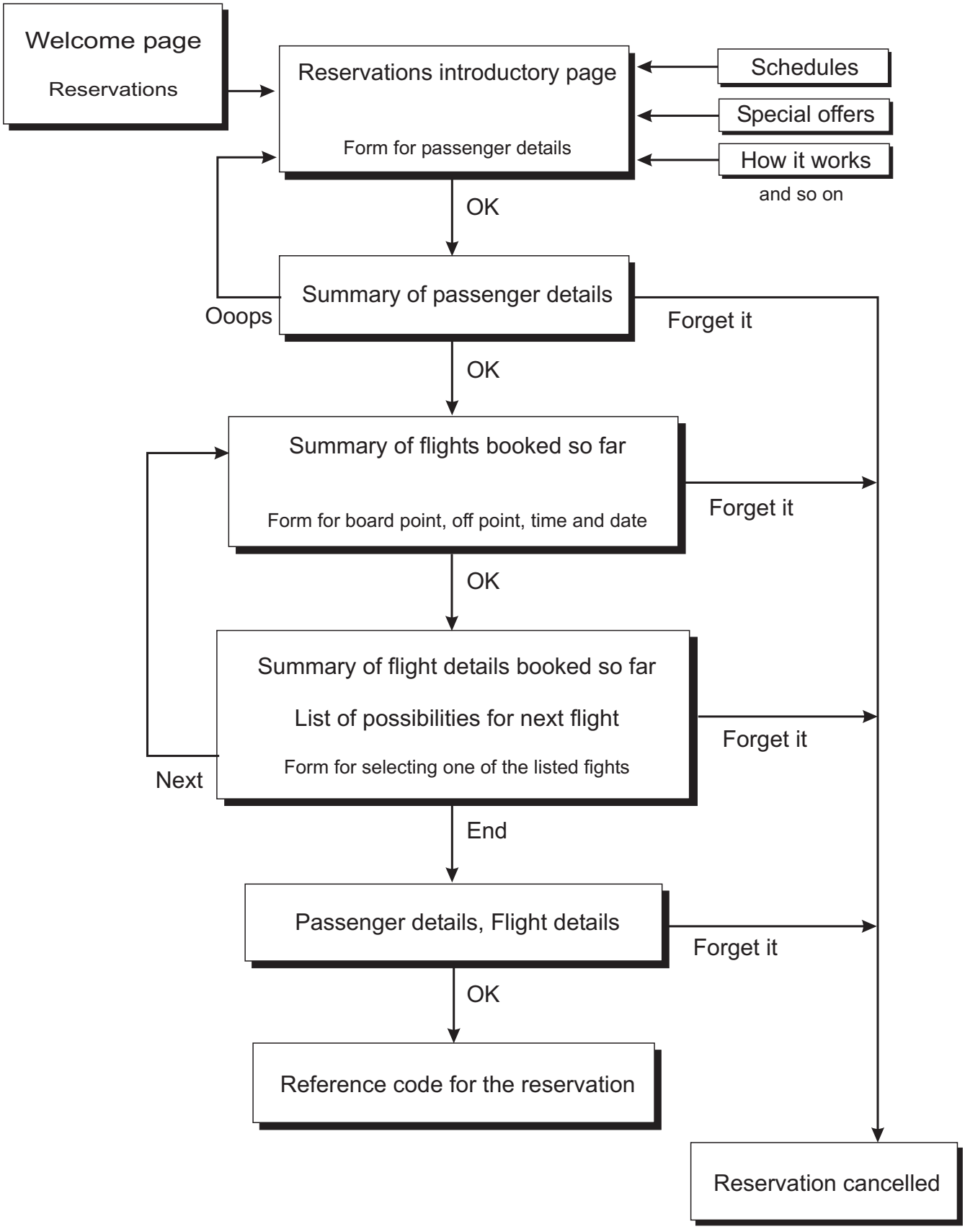


Figure 11. Reserving airline seats – summary

Chapter 5. Creating a Web site on ALCS

This chapter describes how you create a Web site on ALCS. Note that we are not attempting to provide a comprehensive guide to designing and implementing a Web site.

5.1 Getting started

You will need:

- A reference book on HTML
- IBM MVS publications
- Plenty of L3 long-term pool
- TCP/IP support in your MVS system
- ALCS TCP/IP and Web server PTFs
- A work-station with a TCP/IP connection to ALCS
- A Web browser, with viewers for the content-types you intend to use
- A text editor for working on HTML text
- A graphics package, if you want to include images in your pages
- A scanner, if you want to include scanned images in your pages
- A multimedia package and associated hardware for capturing and manipulating sound clips, movie clips, and so on.

5.2 ALCS generations for the Web server

You must run some ALCS generations to configure ALCS to run the Web server. These generations:

- Configure ALCS to include TCP/IP support.
- Specify a nondefault VTAM receive buffer size.
- Define the ALCS application communication resources for the Web server itself and the HFS application.
- Define the TCP/IP communication resource for the Web server.
- Define the WWW sequential file for the access log.

Appendix A, “ALCS generation parameters for the Web server” on page 113 describes the generation macro parameters for these.

Note that the ALCS HFS uses L3 long-term pool to contain:

the HFS directories
files in the HFS
HFS state blocks.

You may want to increase the number of L3 long-term pool records to allow for this.

5.3 Setting-up static Web pages and objects

5.3.1 First steps – your welcome page

Your ALCS Web server will not work until you set up some basic information. To set up this basic information, you must:

1. Create a welcome page.

   We suggest that you do this on a PC. You can use:

   - Your favorite text editor (you could use the OS/2 Enhanced Editor)
   - Your favorite word-processor (you could use Lotus Word Pro)
   - A Web authoring tool (you could use Lotus InterNotes Web Publisher)

   Your first welcome page can be very simple. We discuss this in more detail in 5.3.2, “Creating your first welcome page – very simple example” on page 53 and 5.3.3, “Creating your first welcome page – slightly more exciting” on page 54.

2. Create any images or other multimedia objects for your welcome page.

   We suggest that you do this on a PC. You can use:

   - Your favorite graphics package (you could use Lotus Freelance)
   - Your favorite multimedia package (you could use Lotus Multimedia)

   You may need special equipment, for example a scanner, to “capture” images or multimedia objects from sources such as photographs, and video or audio recordings.

3. Log-on to ALCS with a 3270 emulation session on your work-station. Route the session to the ALCS hierarchical file system (HFS) application with the `ZROUT` command (see 7.1.1, “Routing your terminal” on page 79).

4. Create the `/www` directory in the ALCS HFS (see 7.4, “Creating directories” on page 81).

5. Up-load the images or other multimedia objects for your welcome page.

   Use the `send` command in a DOS or OS/2 session on your work-station, or use the `Transfer` pull-down menu (or equivalent) in your 3270 emulation session (see 7.11, “Up-loading files from a PC” on page 84).

   You *must* up-load these objects into the `/www` directory or a subdirectory of `/www`. The ALCS Web server does not allow Web browsers to get objects from other directories.

6. Up-load your welcome page.

   Use the same method you use for step 5. Note that you must load your welcome page as one of:

   ```
   /www/index.htm
   /www/index.html
   /www/welcome.htm
   /www/welcome.html
   ```

Your ALCS Web server will now respond to “basic” requests from browsers. For example, if the TCP/IP domain name of your server is `mycompany.com`, and if you are using the default port number (80) for your ALCS Web server, then a Web browser can display your welcome page with a URL like:

```
http://mycompany.com
```

5.3.2 Creating your first welcome page – very simple example

Imagine we are “Acme Cricket Equipment Suppliers”. We want to set up a very basic welcome page that simply tells people we exist and plan to have an interesting Web site.

First we create a directory called `C:\WWW` on our PC. This is where we plan to build our Web pages.

Now we use a simple text editor to create our home page. We call it `INDEX.HTM`. It might look something like this:

```
<html>

<head>
<title>Acme Cricket Equipment Suppliers</title>
</head>

<body>
<h1>Acme Cricket Equipment Suppliers</h1>
<p>This site under construction.
Please visit us again soon.
<p>But don't wait, don't hesitate, telephone us <em>now</em> on:
<menu>
<li>Inside UK: 0181 818 4203
<li>Outside UK: +44 181 818 4203
</menu>
</body>

</html>
```

Figure 12. Very simple home page for Acme Cricket Equipment Suppliers

Possibly we are not confident that our HTML will look “right” (it is our first try). Just to be sure, we can start our Web browser and ask it to show us our welcome page. We explain how to do this in 5.3.7, “Testing your pages on a PC – matching the PC and ALCS HFS names” on page 58.

We have no multimedia effects in this (it is our first try), so when we are happy it looks right, we can up-load our welcome page immediately.

We set up a 3270 emulator session with ALCS (session A) and log-on to ALCS. In this session, we route to the ALCS HFS application (HFS1) like this:

```
ZROUT HFS1
```

Now we remember to create the Web server directory and make it our current directory, like this:

```
md /www
cd /www
```

Now we switch to an OS/2 or DOS session and upload our welcome page like this:

```
send c:\www\index.htm a:index.htm ascii
```

Of course, this welcome page is not very exciting. Lets try something slightly more exciting...

5.3.3 Creating your first welcome page – slightly more exciting

Our marketing department tells us they prefer something slightly more exciting than our first try (5.3.2, “Creating your first welcome page – very simple example” on page 53). They want to include our company logo and a background (the background is also the company logo – but in pale colors).

We use a scanner to create two digitized images of our company logo. One is a graphics information file (GIF) in bright colors, and one is a Joint Photographic Experts Group (JPEG) file in pale pastel colors.

We decide to put our images in a separate directory on the ALCS HFS – `/www/images`. To make it easy to test our new page on the PC, we put the images in the PC directory `C:\WWW\IMAGES`.

Now we use our text editor to change our welcome page (C:\WWW\INDEX.HTM) to:

```
<html>

<head>
<title>Acme Cricket Equipment Suppliers</title>
</head>

<body background="images/acmelogo.jpg">

<center><img src="images/acmelogo.gif" alt="Acme Cricket Equipment Suppliers"></center>

<p>This site under construction.
Please visit us again soon.
<p>But don't wait, don't hesitate, telephone us <em>now</em> on:
<menu>
<li>Inside UK: 0181 818 4203
<li>Outside UK: +44 181 818 4203
</menu>
</body>

</html>
```

Figure 13. Slightly more exciting home page for Acme Cricket Equipment Suppliers

Notice that we remember to include a plain text alternative to our graphic company logo (`alt="Acme Cricket Equipment Suppliers"`) in case someone visits our site with “do not load graphics”.

Again, we can preview our new welcome page, see 5.3.7, “Testing your pages on a PC – matching the PC and ALCS HFS names” on page 58.

Once we are satisfied, we can load our new home page onto ALCS.

First, in our 3270 emulator session, we create the `images` directory and make it our current directory, like this:

```
cd /www
md images
cd images
```

and up-load our images (from a DOS or OS/2 session) to the ALCS HFS with:

```
send c:\www\acmelogo.gif a:acmelogo.gif
send c:\www\acmelogo.jpg a:acmelogo.jpg
```

Finally we replace our old welcome page with our new (exciting) welcome page with:

```
send c:\www\index.htm a:index.htm ascii
```

We are careful to load the imbedded images to ALCS *before* we load our new HTML. If we load the HTML first then a browser might fail to load the page correctly (the HTML is there, but the imbedded images it uses are not).

5.3.4 Directory structures and naming conventions

In our example in 5.3.2, “Creating your first welcome page – very simple example” on page 53, we loaded a home page called `index.htm` into `/www` directory of the ALCS HFS. These names have special significance, as follows:

`/www` This directory is the default **server root** for the ALCS Web server.

`index.htm` This file is a **welcome page** for the ALCS Web server.

The following paragraphs explain these special names and some other things you need to know about naming your files and directories. Chapter 6, “HFS file names” on page 75 includes more information on this topic.

The server root

When the ALCS Web server receives a request from a Web browser, it interprets the file-name component of the URL as a request for a file within the server root. For example, the URL:

```
http://www.someplace.com/interesting/information.html
```

requests the file:

```
/www/interesting/information.html
```

Welcome pages

The URL in a request from a Web browser can refer to a directory. If it does, the ALCS Web server interprets the URL as a request for the welcome page in that directory. For example, the URL:

```
http://www.someplace.com/
```

requests the welcome page from the `/www` directory. The URL:

```
http://www.someplace.com/cricket/
```

requests the welcome page from the `/www/cricket` directory.

Note that a trailing `'/'` in a URL forces ALCS to look for the welcome page. For example:

When ALCS receives the URL...	...it interprets the request as:
http://www.someplace.com/cricket	If /www/cricket is a directory, request the welcome page for the directory. If it is not a directory, request the object itself. If it does not exist, ALCS returns “not found”.
http://www.someplace.com/cricket/	If /www/cricket is a directory, request the welcome page for the directory. If it is not a directory or does not exist, ALCS returns “not found”.

Your users will expect you to have a welcome page in the ALCS server root directory. This is the first page they expect to see when they visit your site. You can also have welcome pages in other directories (see 5.3.5, “Adding more pages to your site” on page 57). ALCS Web server welcome pages are files (or programs, see 5.4, “Setting-up dynamic Web pages – interfacing to your application” on page 59) with one of the following names:

1. index.htm
2. index.html
3. welcome.htm
4. welcome.html

When the ALCS Web server receives a request for a welcome page, it searches the directory for these special names. The first file or program that it finds is the welcome page for the directory. For example, if a directory contains:

A directory called index.htm[4]
A file called welcome.htm
A program called welcome.html[5]

then the welcome page is the file called welcome.htm.

Naming imbedded objects

In our example in 5.3.3, “Creating your first welcome page – slightly more exciting” on page 54, we constructed our home page /www/index.htm and imbedded images in it with HTML tags like:

```
<body background="images/acmelogo.jpg">
<img src="images/acmelogo.gif" alt="Acme Cricket Equipment Suppliers">
```

The names images/acmelogo.jpg and images/acmelogo.gif refer to files within the same directory as the file that imbeds them. In this case, the file that imbeds them is in the directory /www, so the names refer to the files /www/images/acmelogo.jpg and /www/images/acmelogo.gif.

You can also specify names that refer to files in the server root directory. These names include an initial '/' character. In our home page, the following two HTML tags are equivalent:

4 We recommend that you do not use names like index.htm for directories.

5 We recommend that you avoid having several things that look like welcome pages in the same directory.

```
<body background="images/acmelogo.jpg">
<body background="/images/acmelogo.jpg">
```

because the file that imbeds `images/acmelogo.jpg` is in the server root.

But if we have a file in one subdirectory of the server root that imbeds objects from another subdirectory, the difference is significant. For example:

If the file...	...includes...	...it imbeds the file:
/www/a/some.htm	<img src="images/acmelogo.gif">	/www/a/images/acmelogo.gif
/www/a/some.htm	<img src="/images/acmelogo.gif">	/www/images/acmelogo.gif

5.3.5 Adding more pages to your site

Adding new static pages to your Web site is very much the same process as setting up your welcome page. You build your new pages and create any images or multimedia objects on your PC and load them onto the ALCS hierarchical file system.

Always load any imbedded objects and files included by server side includes before you load the pages that imbed them; that way you can be sure the pages work correctly for users who browse them.

You also need to include hyperlinks from your existing page or pages to your new pages. Again, be sure to load the new pages before you add hyperlinks from existing pages.

You need to think about how you organize your pages and imbedded objects into directories and subdirectories of the ALCS hierarchical file system. We discuss how you might do this in 4.1.3, “Site structure” on page 36.

5.3.6 Name qualifiers and metainformation

When the ALCS Web server sends a file to a Web browser, it includes metainformation that tells the browser how to interpret the file. In particular, it tells the browser the **content-type** of the file.

The ALCS Web server uses the file name to decide the content-type. It divides the file name into components called **qualifiers**, which are parts of the file-name separated by dot (.) characters. For example, it divides the file name `index.htm` into the qualifiers `index` and `htm`.

By default, ALCS uses the last qualifier to decide the content-type of the file. Figure 14 shows some examples. The full list is in B.1, “File name qualifiers for content type” on page 115.

If you want to use a different naming convention for your files, you must provide an ECB-controlled installation-wide exit program to determine the corresponding content-types. We describe these in 9.1, “Overriding content-type defaults” on page 103. Be aware that you might confuse some Web browsers if you use a “nonstandard” naming convention; the browsers might display the information wrongly.

Figure 14. Decoding the content-type – some examples		
Qualifier	**Content-type**	**Explanation**
`htm`	`text/html`	A Web page in HTML
`gif`	`image/gif`	An image in GIF format
`mpeg`	`video/mpeg`	A movie clip in MPEG format

Note

We hope to implement more sophisticated interpretation of file-name qualifiers. For the moment, we suggest you use names with just two qualifiers, and use the second qualifier to indicate the content-type.

Note that the ALCS Web server does *not* use name qualifiers to associate content-types with directories or programs (see 5.4, “Setting-up dynamic Web pages – interfacing to your application” on page 59).

5.3.7 Testing your pages on a PC – matching the PC and ALCS HFS names

When you build a new Web page, or change one you already have, you probably want to see how it looks on a Web browser *before* you load it onto your Web server. This is especially likely if you use a simple text editor to develop your pages.

A nice easy way to do this is to use a Web browser on the PC where you develop the Web pages. With most Web browsers, you do not need to up-load the files to your server and have your browser request the files from the server. Instead, you can have your Web browser get the files directly from your PC's disks.

For example, on IBM's Web Explorer, you use the `File` pull-down menu and select `Open file...`.

If you want to use your PC Web browser to “test” Web pages, we suggest you use the same names for your Web files and directories on both the PC and the ALCS HFS.

This is what we do in our example in 5.3.3, “Creating your first welcome page – slightly more exciting” on page 54. The file and directory names we use are:

File name on the PC	File name on the ALCS HFS
`C:\WWW\INDEX.HTM`	`/www/index.htm`
`C:\WWW\IMAGES\ACMELOGO.JPG`	`/www/images/acmelogo.jpg`
`C:\WWW\IMAGES\ACMELOGO.GIF`	`/www/images/acmelogo.gif`

Because we use the same file and directory names, we can load the file `C:\WWW\INDEX.HTM` to our Web browser and it “understands” the references to the imbedded objects – it loads them from the PC's disks and we see the page just the same as if we requested it from the Web server.

5.4 Setting-up dynamic Web pages – interfacing to your application

5.4.1 How the ALCS Web server handles static Web pages

When the ALCS Web server receives a request for a static page (like the welcome page in 5.3.2, “Creating your first welcome page – very simple example” on page 53), it processes the request something like this:

1. A component of the Web server, called the **receiver** looks at the request.
2. The receiver calls the HFS support routines to locate the requested file name.

Assuming the file name exists, and is a file:

3. The HFS support routines return information about where the file is.
4. The receiver passes this information to another component of the Web server, called the **sender**.
5. The sender builds the response (which includes both the file itself, and some header information) and sends the response back to the browser.

Figure 15 shows this processing schematically.

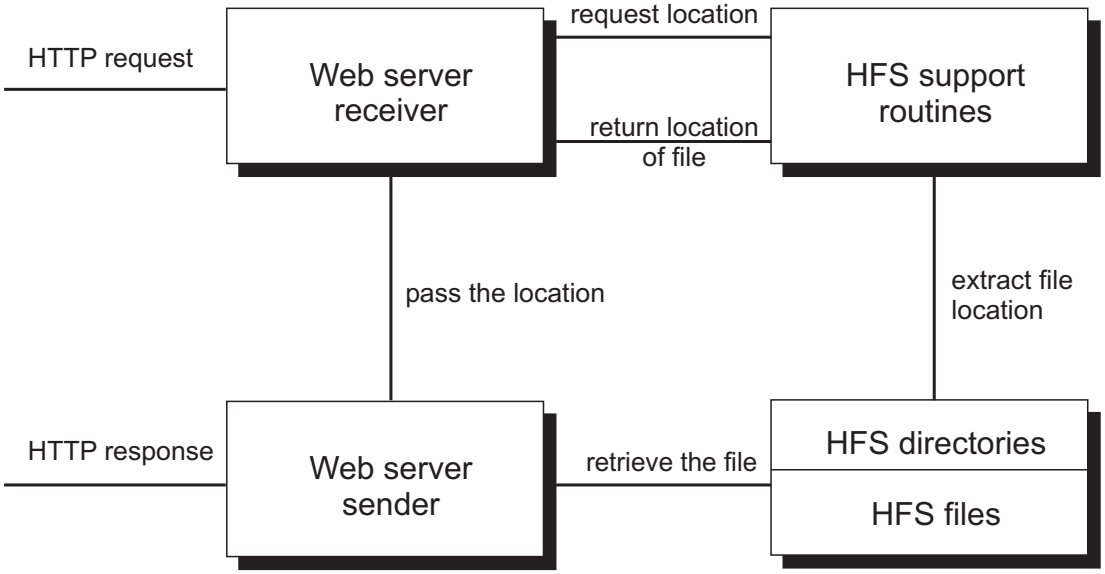


Figure 15. Retrieving static pages

Figure 15 shows the ALCS Web server processing a request that completes successfully. This assumes that there is no error in the request, that the file exists, and so on. Of course we can not guarantee that everything is correct.

If something goes wrong, the Web server receiver program does not pass the file location to the sender. Instead, it passes an error code. The Web server sender constructs and sends an appropriate error response. Figure 16 on page 60 shows an example – the requested file does not exist.

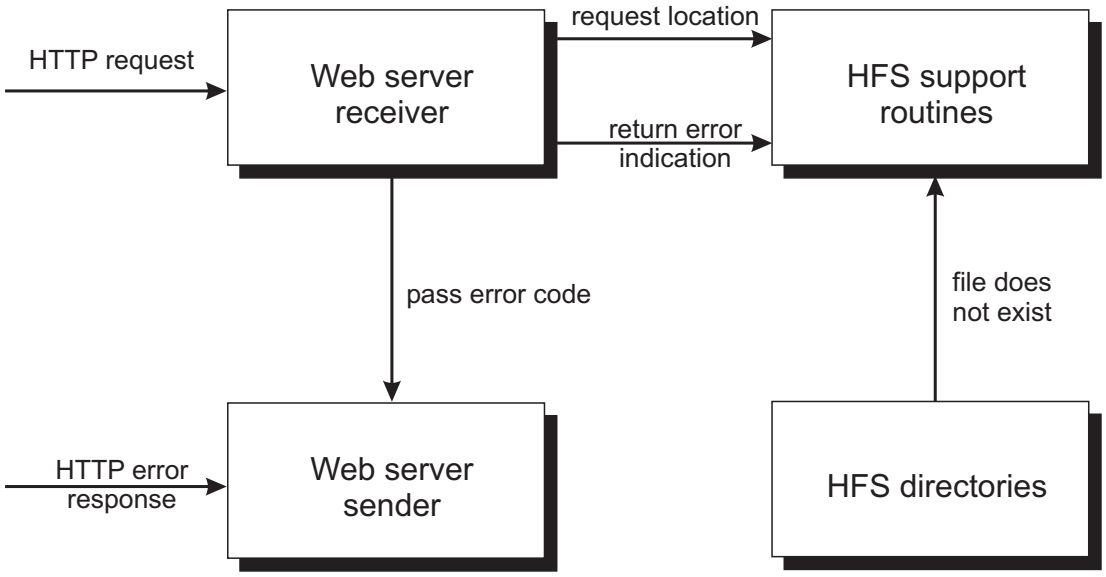


Figure 16. Retrieving static pages – error processing

5.4.2 How the ALCS Web server handles dynamic Web pages

As you would expect, dynamic Web pages are slightly more complex. The file name refers to an ECB-controlled application program (called a **Web program**) instead of to a file. In this case, the Web server receiver processes the request something like this:

1. The Web server receiver looks at the request.
2. The receiver calls the HFS support routines to locate the requested file name.

Assuming the file name exists, and refers to an application program (a Web program):

3. The HFS support routines return information about the Web program.
4. The receiver calls the Web program.
5. The Web program builds the response file and passes it to the Web server sender.
6. The sender adds header information and sends the response back to the browser.

Of course, your Web program may not like the request. In this case, it can pass an error code to the Web server sender.

Figure 17 on page 61 shows this processing schematically.

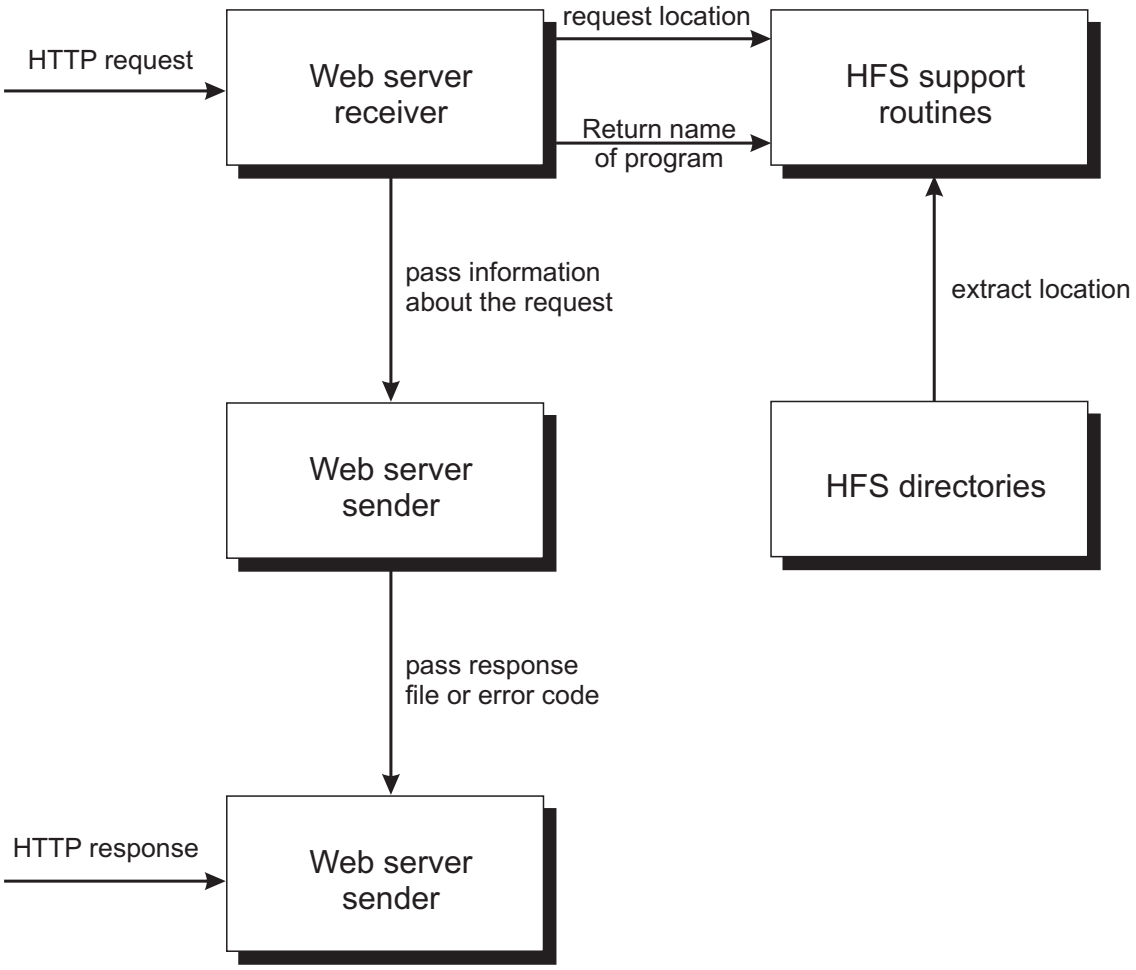


Figure 17. Retrieving dynamic pages

You may be wondering how the HFS directories can associate an HFS file name with an ECB-controlled application program. Many hierarchical file systems store executable programs as files. But ALCS does not do this. Instead, ALCS allows you to define an HFS name for an ECB-controlled program using a special command (see 7.5, “Creating and changing programs” on page 81).

5.4.3 Interfacing Web programs to your existing application

When you look at Figure 17, you may wonder how your “Web application” interfaces to your existing application.

The following sections describe some ways to do this.

Developing new transactions

One way to interface to your existing applications is to develop your Web application the same way you develop support for new transactions.

Your Web application can access your existing application data base in the same way as any other ECB-controlled application program. It can also call (ENTER/BACK) existing application programs. Figure 18 on page 62 shows this schematically.

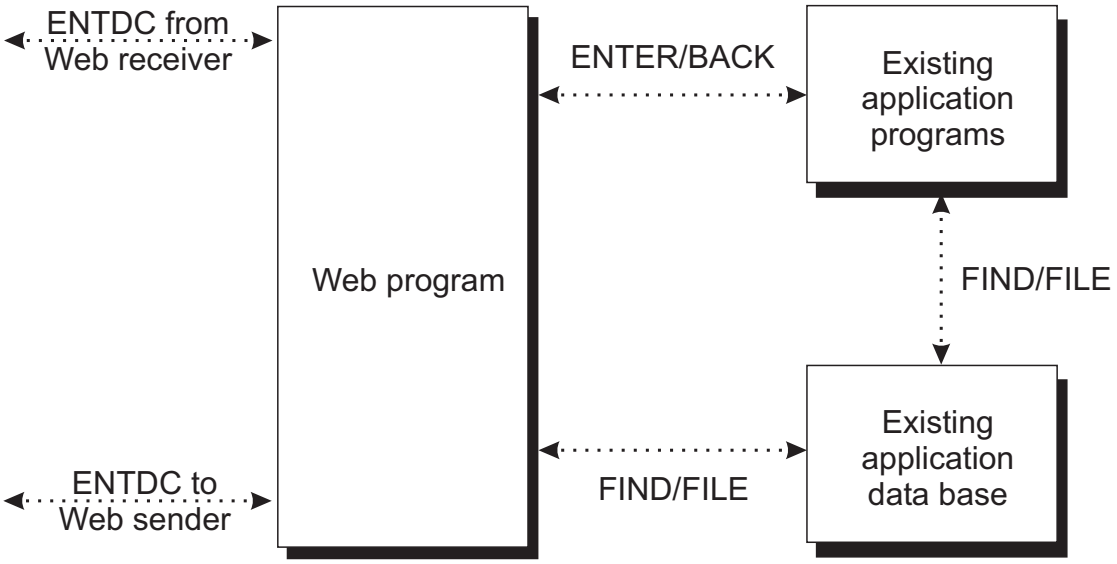


Figure 18. Web program – new transaction

Using existing transactions – your application structure

Of course, if your existing application already supports the transactions you want to implement on the Web, you may prefer not to develop new transactions. For example:

- You may not have programmers available to develop new transactions. (They may be busy doing other things.)
- You may not have enough time to develop new transactions. (You may want to get your transactions onto the Web very quickly.)

Exactly how you implement an existing transaction on the Web depends on how your existing application is structured.

If your existing transactions are only available to terminals, your application structure may be something like Figure 19. (The IPARS airline reservations application that we use in IBM to test ALCS is very much like this.)

Figure 19. Application structure – simple case

If your existing transactions are available to things other than terminals then your application structure may be more complex. For example, some Airline reservation systems process transactions from a variety of sources, including

- Terminals, such as IBM 3270s
- Global distribution systems (GDSs)

- Other reservation systems, using host-to-host (HTH) links
- and so on.

Figure 20 shows how such an application might be structured.

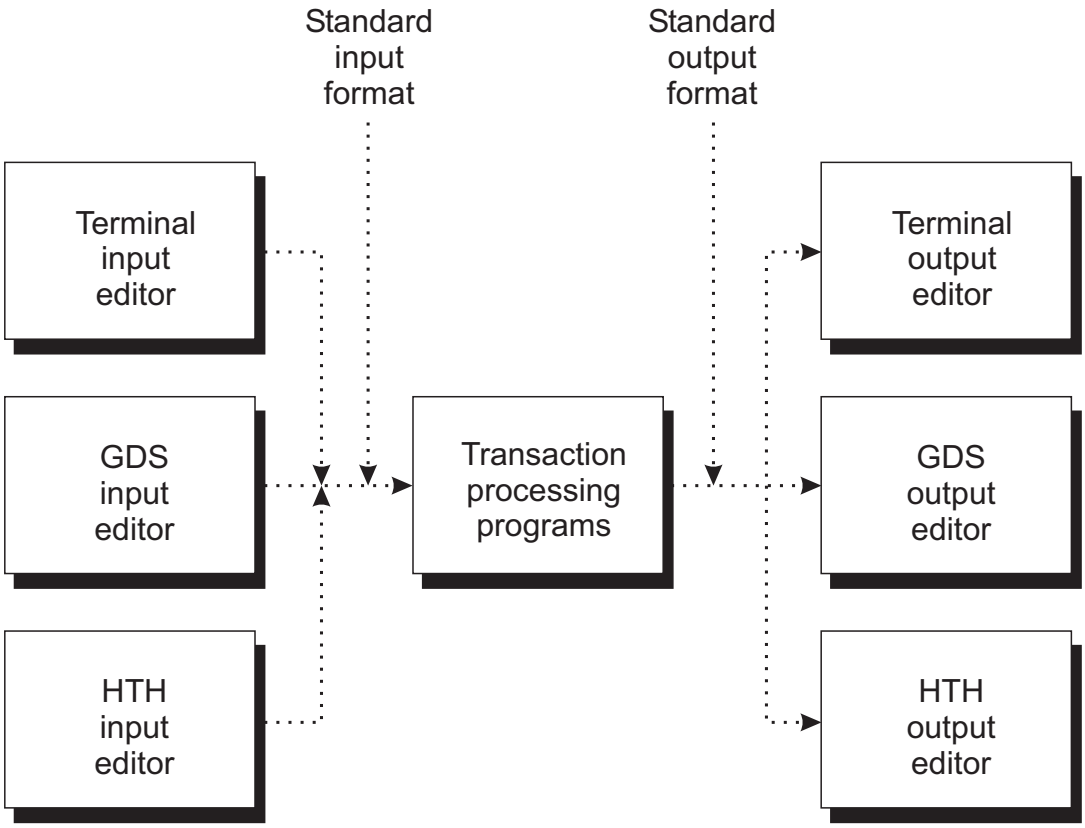


Figure 20. Application structure – more complex case

Using existing transactions – special input and output editors

If your application structure is similar to Figure 20, you can easily interface to the Web server by developing new Web input and output message editors. Figure 21 on page 64 shows this schematically.

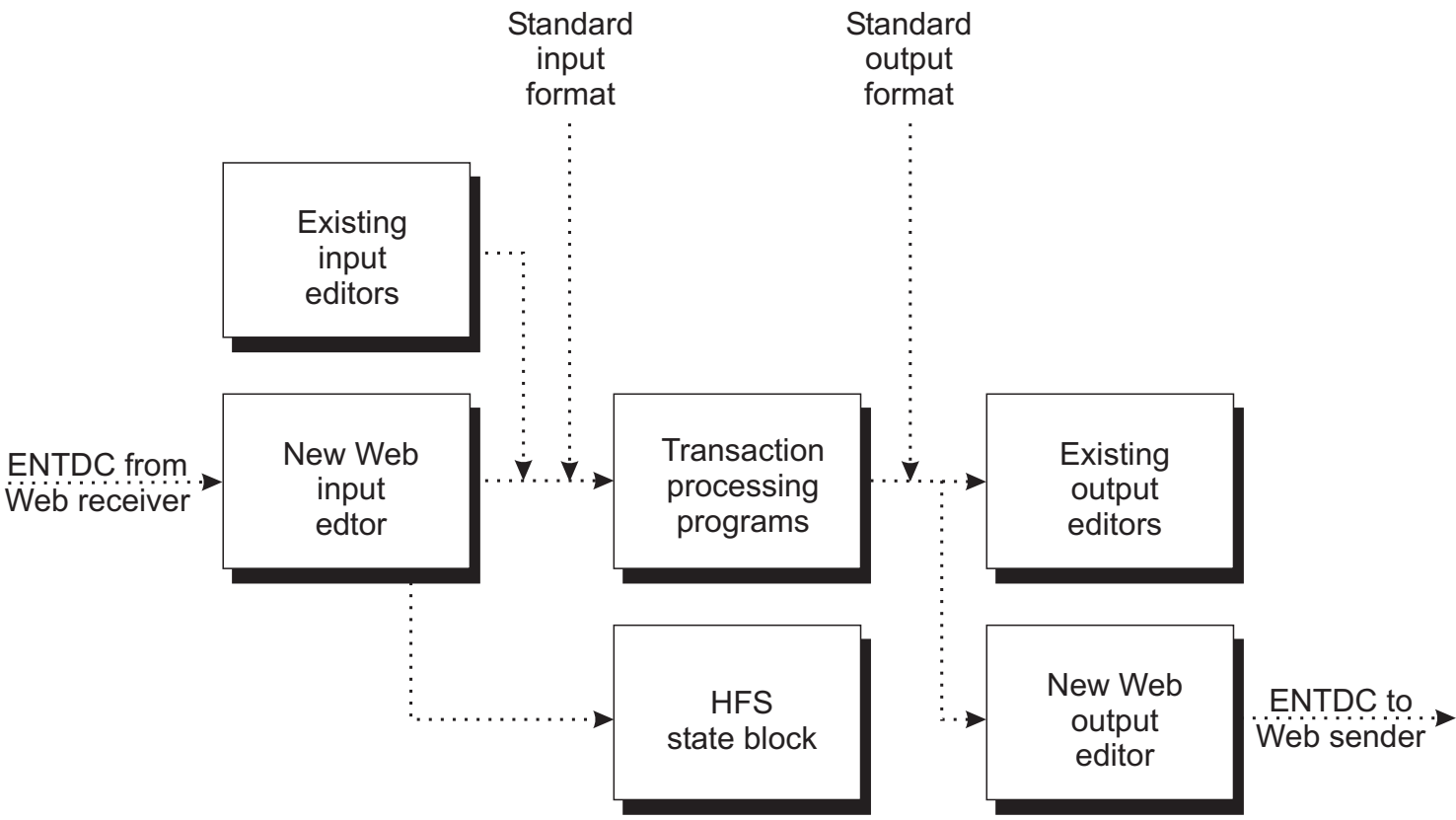


Figure 21. Web program – using new input and output editors

You will notice that in Figure 21, we include a component called the **HFS state block**. The HFS state block is a DASD record that you can use to pass information from your Web input message editor to your Web output message editor.

The Web server receiver program saves some information in the HFS state block (and files it, just in case your Web input editor does not). Then it passes the HFS state block to your Web input editor (attached on level D3 of the ECB). Your Web input editor can save extra information in the HFS state block and file it.

Your Web output editor can read the HFS state block (the ALCS Web server includes a utility program that makes this easy). In fact, even if your Web output editor does not need to use information from the HFS state block, it *must* read the record because the Web sender expects the HFS state block to be attached on level D3 of the ECB when you call it.

We have more to say about the HFS state block in 5.5, “Preserving state information” on page 68.

5.4.4 How we interface the ALCS Web server to our IPARS application

If your application structure is more like Figure 19 on page 62, then you may want to use a technique similar to the one we developed to interface to our IPARS airline reservations application. To help you do this, the ALCS Web server includes a specialized interface program, and we also provide you with the actual application programs we use.

Input processing

The IPARS application includes an input message editor called UII (it is actually two programs, UII1 and UII2).

UII does all kinds of things, many of these we do not need for our Web server transactions. For example, UII handles *all* IPARS reservations transactions and we only want to implement a few of these on the Web – we do not want our Web users to change our schedules, cancel our flights, and so on.

Also, the IPARS UII expects input in the standard ALCS IMSG format. It does not change the input messages significantly because the IPARS transaction processing programs also expect input in IMSG format. Of course, input from the Web browser is *not* in IMSG format.

So we wrote a new Web input message editor. Our new Web input message editor is much simpler than the IPARS UII, even though it does things that UII does not do, including:

- It converts input from the Web browser to IMSG format.
- It creates and initializes a special record called the agent assembly area (AAA) that IPARS transaction processing programs need.

  We have more to say about the AAA in 5.5, “Preserving state information” on page 68.
- It saves information in the HFS state block that we use later when we construct the Web response.

Figure 22 on page 66 shows our input processing schematically. It is similar to the input processing in Figure 21 on page 64.

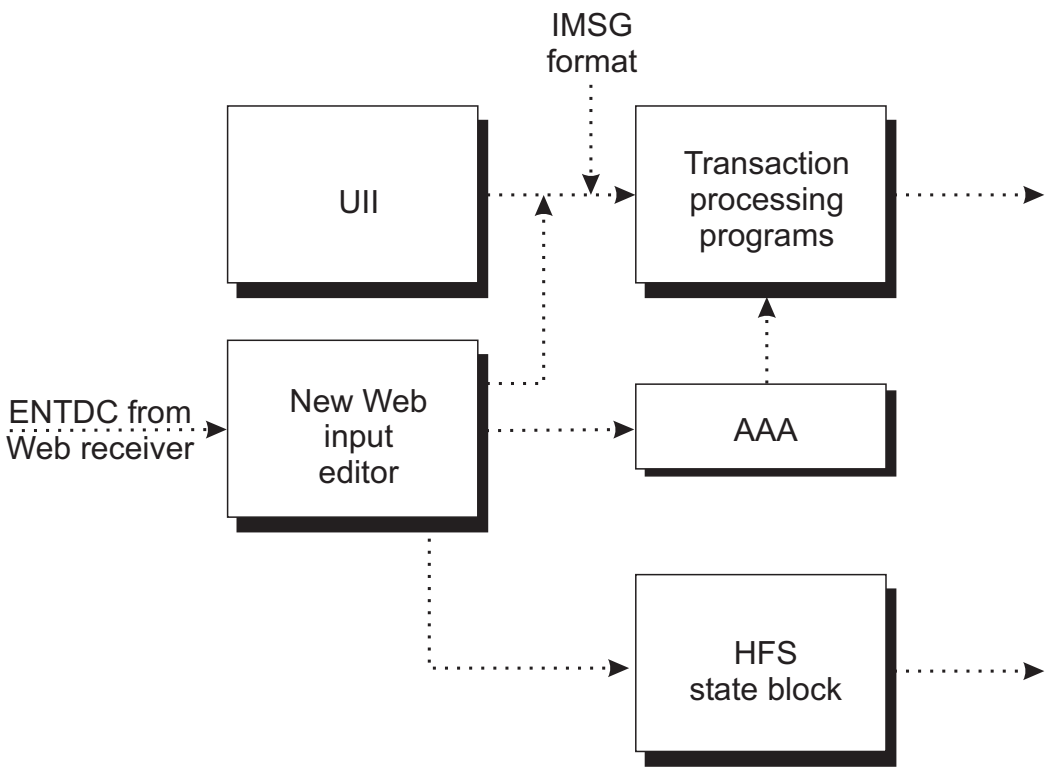


Figure 22. Web interface to IPARS – input processing

Output processing

The IPARS application includes an output message editor called UIO (it is actually three programs, UIO1, UIO2, and UIO3). Unfortunately it is not so easy to develop a special UIO for the Web. In particular:

- UIO is much more complicated than UII.
- Some applications that call UIO expect UIO to return.

We do not wish to do any further development work on the IPARS application. It would be very difficult to provide a new output editor that would work correctly with the rest of IPARS.

Instead, we found the places where UIO does things like:

```
IF (originator is a display) /* this test uses the COMIC macro */
   THEN do one thing
   ELSE do something else
```

and changed them to:

```
IF (originator is a display or the Web)
   THEN do one thing
   ELSE do something else
```

We also found the places where UIO sends (SENDC) the response message and changed them to:

```
IF (originator is the Web) /* we use the COMIC macro to test this */
   THEN CREMC to the Web output interceptor, pass the message block(s)
   ELSE SENDC the response message block(s)
```

The Web **output interceptor** is a component of the ALCS Web server. It reads the HFS state and calls a Web output editor.

The Web ouput editor uses information in the output message block(s) to construct the Web response, and then passes that response to the Web server sender. Output blocks from UIO are in the ALCS OMSG format.

Figure 23 shows this processing schematically.

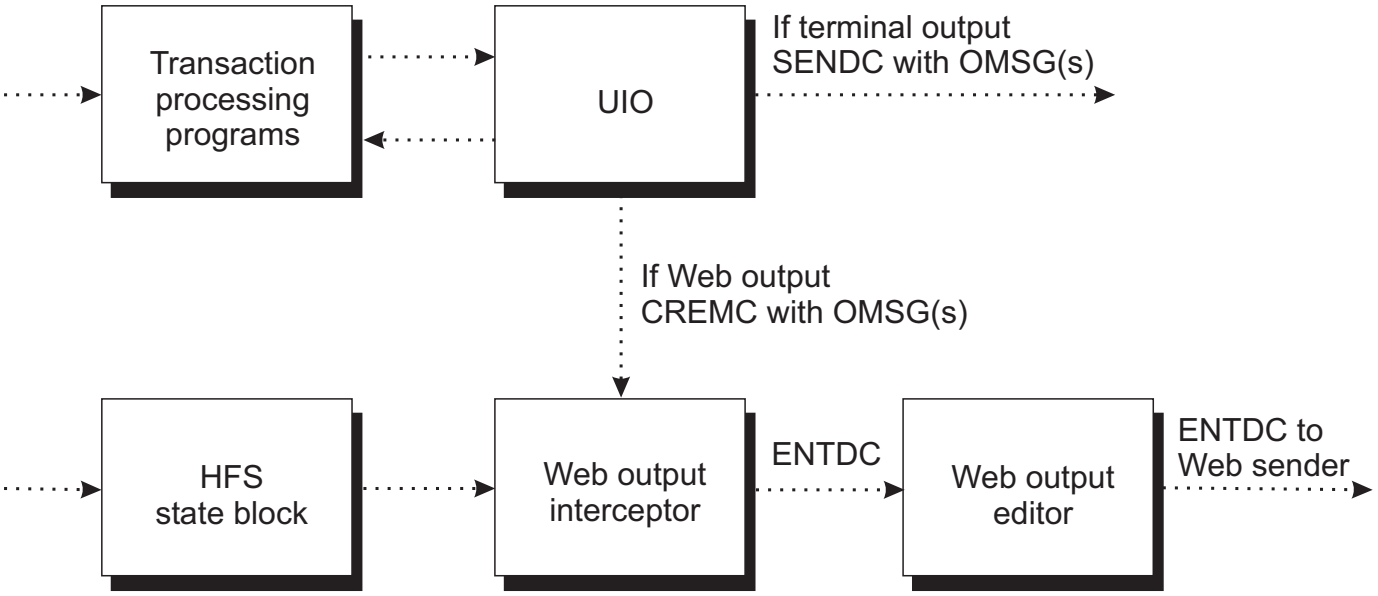


Figure 23. Web interface to IPARS – output processing

The Web output editor may need to know something about the input transaction to help it transform the output OMSGs into Web responses. The HFS state block and the Web output interceptor can help with this.

The Web input editor can save information about the input transaction in the HFS state block so that the Web output editor can use it.

To make life easier (we try to be helpful), our Web output interceptor is designed to work cooperatively with your Web input editor. You can have several Web output editors, each specializing in formatting the response to a particular input transaction. It works like this:

1. Your Web input editor knows which input transaction it is handling. It decides which of your Web output editors is appropriate and saves the name of the Web output editor in the HFS state block.
2. Our Web output interceptor reads the HFS state block, gets the name of your Web output editor, and calls (ENTDC) it.

If your Web input editor does not save the name of a Web output editor, our Web output interceptor sends the OMSG format response back to the Web browser without reformatting.

5.4.5 How the ALCS Web server handles system errors

It is possible (we hope it does not happen) that your application programs contain system errors that prevent normal processing of input messages. ALCS recognizes two types of system error:

- Operational (OPR) error. The application detects the error and issues SYSRA or SERRC.
- Control (CTL) error. The ALCS monitor, or a monitor installation-wide exit, detects the error.

For both types of error ALCS either continues processing the entry (system error with return) or terminates the entry (system error with exit).

For system error with exit, ALCS checks if the application has sent a response to the originator. If the application has *not* sent a response to the originator then ALCS constructs and sends a special response message which tells the end user that their input message did not work. For Web inputs, ALCS sends a special HTTP response code of 500 (internal server error, see Appendix D, “HTTP response codes” on page 121).

5.5 Preserving state information

In a typical transaction processing system, there are a number of cases where your application needs to “remember” what happened before. For example, a sporting goods vendor might have an application that supports a dialog something like this:

```
 Customer:  What cricket bats do you have?
Application: The "Infant" made of attractive red and yellow plastic
             The "Child" made of real wood
             The "Teenager" which is like the "child" only bigger
             The "Tradition" made of finest willow
             The "High-tech" made of titanium and carbon fibre
 Customer:  I'd like to order the "Infant" please
Application: A wise choice, one of our most popular models.
             Please tell us your name
 Customer:  Mr W  G  Grace
Application: Thank you for placing an order, Mr W  G  Grace.
             Please tell us your address
```

and so on.

The application needs to remember that it presented a list of cricket bats so that it can understand what Mr Grace means when he asks for the “Infant”. (There might be “Infant” models of other things; stumps, pads, and so on.) This is an example of **state information**.

5.5.1 Levels of state information

A transaction processing application typically needs at least three levels of state information, including:

- **Message state information**

  This is information that the application keeps while it is processing a single input message.

It includes things like where the input message came from (which we need when we send the response), what the input message requested (which we may need to check from time to time while we are processing the request), and so on.

- **Conversation state information**

  This is information that the application keeps while it is processing a sequence of input messages and responses. In our sporting goods example, the sequence places an order for a cricket bat. In airline reservations applications, a sequence like this is often called a transaction.

  Conversation state information includes things like where we are in the conversation (is it time to ask the customer's name?), and the information we have accumulated during the conversation (the customer wants an “Infant” cricket bat, their name is Mr W G Grace, and so on).

- **Business transaction state information**

  This is information that the application keeps after a complete sequence of input messages and responses. In our sporting goods example, the application saves the entire order information when the order is complete.

  The application may need to refer to (and modify) this information later. For example, the customer may use a new sequence of input messages and responses to change or cancel their order, the shipping department may use a new sequence of input messages and responses to find out what to package and who to send it to (and to record that the cricket bat has been shipped), and so on.

5.5.2 Preserving message state information

In the ALCS Web server, we keep message state information in the HFS state block. This includes things like the HTTP version in the request (which we need because we must use a compatible version for the response).

You can also use the HFS state block to keep your own message state information. For example, if you interface to your application in a similar way to the way we interface to IPARS, you may want to remember information about the input message that you use later to help reformat the response.

Chapter 8, “HTTP application interfaces” on page 87 explains how you can use the HFS state block to save your own message state information and how you can access the information that we save in it.

ALCS associates an HFS state block (actually a real-time data base record) with the communication resource identifier (CRI) of the HTTP input message. But you need to be aware that for HTTP input messages there is no fixed relationship between a CRI and a user agent.

ALCS allocates a “spare” TCP/IP CRI when a client establishes a connection. The CRI is associated with the connection. ALCS “releases” the CRI when it breaks the connection. (You may find it helpful to refer to Figure 3 on page 4.)

There is no guarantee that ALCS will allocate the same CRI the next time the same client establishes a connection. This means you can not save conversation state information in the HFS state block.

5.5.3 Preserving conversation state information – the problem

Most ALCS applications preserve conversation state information in a real-time data base record associated with the terminal (that is, with the terminal CRI). For example, the IPARS application uses a fixed-file record called the agent assembly area (AAA). A special application program called WGR knows how to find the AAA for any given CRI.

This works well with conventional terminal communication because the same terminal always has the same CRI[6] and different terminals always have different CRIs. But it does not work well with Web conversations. In fact it does not work at all because two consecutive requests from the same client can have different CRIs (and two consecutive requests from different clients can have the same CRI).

5.5.4 Preserving conversation state information – tokens

As you might guess, the problem of preserving conversation state information is not unique to the ALCS Web server; most transaction processing platforms have the same problem. And as you might guess, there is a solution. It works like this:

1. When a client starts a conversation (when you receive the first request), you choose a real-time data base record where you can save information about the conversation. We call this the “conversation state record”.
2. You create a “token” that identifies the conversation state record.
3. You save information about the request in the conversation state record.
4. You send the response and include your token.
5. In the next request, the client includes the token. You can use the token to retrieve the conversation state record.

Steps 3 through 5 repeat, using the same token until:

6. The client sends a request that ends the conversation.
7. You use the information collected in the conversation state record to build the complete business transaction (order for goods, flight or hotel room booking, or whatever) and add the business transaction into your data base. You clear the conversation state record so that it can be reused and discard the token.
8. You send the final response. Probably you include a reference number for the business transaction in your response.

You probably have some questions about this solution, including:

- How do I chose a real-time data base record for the conversation state?
- How do I construct the “token”?
- How do I ensure that the client sends my token in each request?
- What do I do if the client never completes the conversation?

We answer these questions in the following paragraphs.

---

[6] It is more accurate to say that the same terminal has the same CRI for the duration of a session. Terminals that connect to ALCS using Telnet may have one CRI for one session and a different CRI for another session. In this context, a conversation is the same as a session so that you *can* use a record associated with the CRI to save conversation state information.

Choosing real-time data base records for the conversation state

As we mentioned in 5.5.3, “Preserving conversation state information – the problem” on page 70, most ALCS applications preserve conversation state information in a real-time data base record associated with the originator CRI. For example, the IPARS application uses the AAA.

For Web conversations, this does not work; you need a way to allocate records dynamically. That is, you allocate a “free” record when a conversation starts, and release it (for reuse) when the conversation ends.

If your application already allocates conversation state records dynamically, you can probably skip straight to “Constructing conversation state tokens” on page 72.

One way to implement dynamic allocation is to use ALCS's long-term pool management. You use GETFC (`getfc`) to allocate a record and RELFC (`relfc`) to release it. Unfortunately, this method has a number of disadvantages, including:

- It can use up long-term pool very quickly.

  Each new conversation uses a long-term pool record and you do not recover the records until you run Recoup.

- If a client abandons a conversation part way through, the long-term pool record never gets released.

To avoid these problems, you can use a fixed-file record to store a table of long-term pool file addresses. Associated with each file address, you also need an “in-use/free” indicator (and some other stuff, see “Constructing conversation state tokens” on page 72 and “Handling incomplete conversations” on page 73). Now your allocate logic looks something like this:

```
Read the fixed-file record
Search the table
IF (there is a free file address)
  Flag the file address in-use
ELSE (there is no free file address)
  Add a new table entry
  Get a new file address (GETFC)
  Store the new file address in the new table entry
  Flag the file address in-use
ENDIF
Write the updated fixed-file record
Initialize the pool record and use it
```

And your release logic looks something like:

```
Read the fixed-file record
Flag the file address free
Write the updated fixed-file record
```

Of course, you must add a Recoup descriptor for the fixed file record.

If you have the IBM TPFDF product installed, it is probably easier to use a TPFDF file with one logical record for each “table entry”.

Using dynamic conversation state records with your existing application

Assuming that your existing application uses a fixed-file record associated with the originator CRI, you need a way to have it use your new dynamic conversation state record. There are essentially two possibilities:

- Pass the pool record to your application. This works if your application keeps the conversation state in memory, and does not attempt to compute its file address from the originator CRI.

  This method works fine with our IPARS application.

- Copy the pool record contents into the fixed-file record associated with the originator CRI and call your application. When your application completes processing, copy the fixed-file record contents back into the pool record.

Constructing conversation state tokens

A very simple way to construct a token for the conversation state is to use the file address itself, converted to a printable hexadecimal or decimal representation. Equally simple is to use the table entry number (or TPFDF logical record number) from your allocation table (see “Choosing real-time data base records for the conversation state” on page 71).

A possible disadvantage of these is that you use the same token again when you use the same record again. This can be a problem if a client starts a conversation and then pauses for a very long time (perhaps hours or even days). Eventually you decide to forget that conversation and reuse the record for another conversation. If the first client now “wakes up” and continues the conversation, you have *two* conversations using the same state record.

One way to avoid this problem is to combine the long-term pool file address (or table entry number, or whatever) with, for example, the time of day (TOD) clock value. This allows you to have a different token for each conversation even when the conversations use the same state record.

However you allocate the token, you probably want to keep it in the table entry (or TPFDF logical record) from your allocation table (see “Choosing real-time data base records for the conversation state” on page 71).

Ensuring that clients include tokens in each request

At the start of 5.5.4, “Preserving conversation state information – tokens” on page 70, we describe a process that includes step 4 “You send the response and include your token” and step 5 “In the next request, the client includes the token”.

There are several ways that you can include your token in a response and ensure that the next request includes the token. These include:

Hidden fields You can put any information you want into a hidden field (actually you can have more than one hidden field in a form, if you wish). When the user submits the form information, the client (the Web browser) includes your hidden fields along with the input that the user provides.

For example, if you include:

```
<input type=hidden name=token value="B74XW1234">
```

then the input from the form will include (along with input the user provides):

```
token=B74XW1234
```

URL queries You can put any information you want into a URL query (actually you can have more than one query on a URL, if you wish).

For example, you can include:

```
<form method=POST action="/myprog.cgi?token=B74XW1234">
```

Note that you can include queries on hyperlink URLs. For more information about HTTP URLs, see Appendix C, “HTTP uniform resource locators” on page 117.

Handling incomplete conversations

At the start of 5.5.4, “Preserving conversation state information – tokens” on page 70, we describe a process that includes step 6 “The client sends a request that ends the conversation”.

If the conversation is ordering some product or service, the conversation might end with your application sending a page that includes:

- A summary of the order information
- A submit button that confirms the order
- A submit button that cancels the order.

The user ends the conversation by pressing one of the two submit buttons. Your application receives input, as follows:

confirm Your application uses the information collected in the conversation state record to build the complete business transaction and adds the completed transaction into your data base.

cancel Your application discards the information collected in the conversation state record.

Note that some applications update the data base before the end of a conversation. For example, many airline seat reservation applications update flight inventory information (how many seats are left on the flight) during the conversation. If your application is like this, it may need to undo (back-out) data base changes when the user selects the cancel option (airlines call this “ignore transaction”).

In either case, your application clears the conversation state record so that it can be reused and discards the token.

Unfortunately there is no way to force the user to press either of the two submit buttons; your application may *never* receive a request that ends the conversation.

A way to deal with this situation is to have a program that checks all the Web conversation state records at regular intervals (say every hour). This type of program is often called a “police program” or a “time-initiated function”. If the program finds a conversation where the last user input is older than a certain amount of time (say four hours), it cancels the conversation.

You may want to make life easier for your police program by including the time of the last user input in the table entry (or TPFDF logical record) from your allocation table (see “Choosing real-time data base records for the conversation state” on page 71).

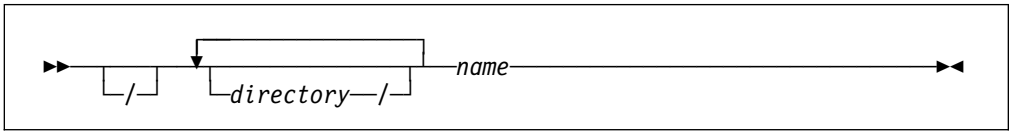
5.5.5 Preserving business transaction state information

Your existing ALCS application almost certainly already includes a way to preserve business transaction state information. For example, the IPARS airline reservation system preserves business transaction state information in the passenger name record (PNR).

Web transactions do not affect how your application does this, except that you may want to tell your user a code that they can use to refer to the transaction later. This code might be an invoice number. The IPARS airline reservation system uses a code called the PNR locator.

Chapter 6. HFS file names

An ALCS hierarchical file system file name is a character string of up to 1023 characters (8-bit binary values). The format is:

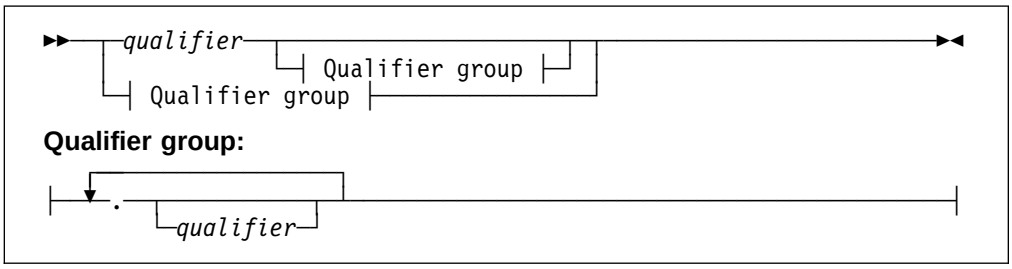


Where:

directory
Name of a directory.

name
Name of a file or program object.

The name of a directory, or of a file or program object, is a string of up to 255 characters. It can include any character (8-bit binary value) except X'00' (but see 6.2, “Special characters in HFS file names” on page 76). These names have the following syntax:



Where:

qualifier
Any character string that does not include a dot (.) or null (X'00').

Qualifiers that follow the first dot are often called file-name **extensions**.

The names that comprise a single dot (.) and a double dot (..) have special meanings, see 6.1, “Absolute and relative file names”.

6.1 Absolute and relative file names

The ALCS hierarchical file system distinguishes between **relative** file names and **absolute** file names.

Absolute file names start with a '/' character. The initial '/' indicates the root directory. For example, the name:

```
/www/figures/fig1.jpeg
```

refers to the file `fig1.jpeg`, which is in the directory `figures`, which is in the directory `www` in the root.

Relative file names do not start with '/'. They refer to the current directory. For example, the name:

```
figures/fig1.jpeg
```

refers to the file fig1.jpeg, which is in the directory figures, which is in the current directory. If the current directory is /www then the following two file names refer to the same file:

```
figures/fig1.jpeg
/www/figures/fig1.jpeg
```

The special name that consists of a single dot (.) refers to the current directory. For example, the following two file names refer to the same file:

```
figures/fig1.jpeg
./figures/fig1.jpeg
```

The special name that consists of a double dot (..) refers to the parent of the current directory. For example, if the current directory is /www, the file name:

```
../figures/fig1.jpeg
```

refers to the file fig1.jpeg, which is in the directory figures, which is in the root.

ALCS only allows you to use the single or double dot at the start of a file name. For example, ALCS does not allow the following file names:

```
/../figures/fig1.jpeg
../..
/figures/./fig1.jpeg
```

6.2 Special characters in HFS file names

ALCS file names can contain any characters (in this context, a character is any 8-bit combination), except for the null (X'00').

File names are case-sensitive, so that the names This and this are *not* the same. (Because ALCS HFS commands translate all file names to lower case, see 7.1.2, “Case sensitivity in HFS commands” on page 79, you can not create file names that include upper-case characters.)

We strongly advise you to avoid the following characters in file names:

Wild cards We reserve the star (*) and query (?) characters for possible use as wild cards (see 7.1.3, “Wild-card characters in commands” on page 79).

Slashes ALCS uses the forward slash (/) for delimiting directory names. We reserve the backward slash (\) for possible use in extensions to the ALCS HFS commands.

Quotes We reserve the single quote (') and double quote (") for possible use in extensions to the ALCS HFS commands.

Brackets We reserve the left bracket ([) and right bracket (]) for possible use in extensions to the ALCS HFS commands.

Minus We recommend that you do not create names with an initial minus (-) character. We reserve this use of the minus for possible use in extensions to the ALCS HFS commands. However you *can* use the

minus in other positions in file names. For example, you should avoid calling a file `-myname`, but the name `my-name` is OK.

Upper-case ALCS translates all file names that you include in these commands to lower case. This means that if you enter the file name `/WWW/INDEX.HTM`, ALCS uses the file name `/www/index.htm`.

We recommend that you always type file names in lower case in ALCS commands.

6.3 File names in HTTP URLs

If you are creating names that might appear in HTTP URLs, you should try to stick to simple names that contain only:

Lower case alphabetics (a-z)
Numerics (0-9)
Dot (.) and minus (-)

and avoid using either the dot or the minus as the first character of the name.

Other characters in URLs can cause problems for some users. For example:

- ASCII characters in the ranges X'02' through X'1F' and X'80' through X'FF' are often not represented on keyboards.
- ASCII characters in the range X'80' through X'FF' may not be transmitted correctly across some TCP/IP networks.
- The following characters have special significance in HTTP URLs:

  Ampersand (&)
  Hash or pound (#)
  Percent (%)
  Question-mark or query (?)
  Semicolon (;)

- Currency symbols often appear different on different equipment. What shows on a screen (or keyboard) in the UK as `'£'` may show as `'$'` in the USA or `'¥'` in Japan.

These characters *can* appear in file names in HTTP URLs, but users may be forced to use **escape sequences** to enter them on their Web browser. An escape sequence in an HTTP URL comprises a percent (%) followed by the hexadecimal code for the ASCII character. For example, to request the file `weird;name.html`, you use `weird%3Bname.html` (X'3B' is the ASCII code for the semicolon).

For more details about HTTP URLs, see Appendix C, “HTTP uniform resource locators” on page 117.

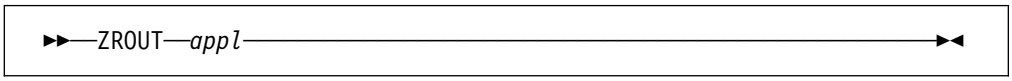
Chapter 7. HFS and PC file transfer commands

This chapter describes the syntax of the commands (input messages) for the ALCS hierarchical file system application, including the PC file transfer command.

7.1 Using HFS and PC file transfer commands

7.1.1 Routing your terminal

To use any of the commands described in this section, you must route your terminal to the ALCS HFS application. To do this, use the ALCS `ZROUT` command, as follows:



Where:

appl
ALCS application name (CRN) of the ALCS HFS application. This is the name that you specify in your ALCS communication generation.

While your terminal is routed to the ALCS HFS application, you can also use other ALCS commands.

7.1.2 Case sensitivity in HFS commands

ALCS ignores the case that you use when you enter the commands in this chapter. This means that ALCS treats `this`, `This`, and `tHiS` exactly the same.

When a file name appears in a command, the command translates the file name to lower case. This means you can not use ALCS HFS commands to refer to HFS files that include upper-case characters.

7.1.3 Wild-card characters in commands

Wild-card characters (or just “wild cards”) are special characters that you can include in commands so that the command acts on several files with similar names.

The ALCS HFS commands that support wild cards allow the HFS name to contain any number of star (*) characters and question mark (?) characters in any position.

You can think of the star as meaning: “any string of characters” and a question mark as “exactly one character” For example, the command `dir abc.*` would include all the following files (if they exist):

```
abc.
abc.def
abc.d.e.f
abc.hello
```

but it would *not* include:

```
abc
```

```
abcdef
ab.cdef
```

The command `dir a*c.???` would include:

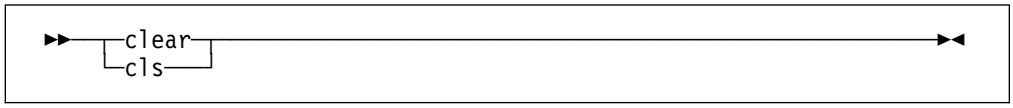
```
ac.pgm
abc.def
```

but it would *not* include:

```
abcd.efg
abc.d
```

7.2 Clearing the screen

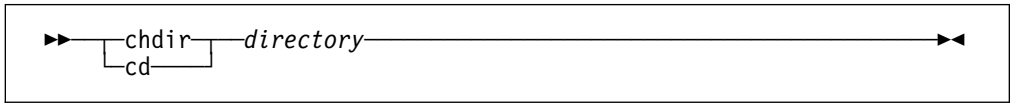
Use the following to clear your terminal screen:



These commands have no parameters.

7.3 Changing the current directory

Use the following to change the current directory for your terminal:



Where:

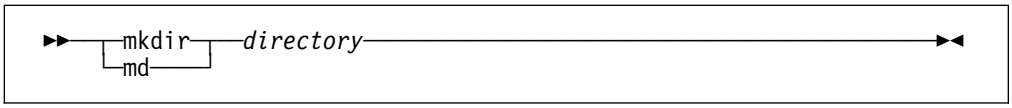
directory
Name of the directory you want as your current directory. This name can not contain wild cards.

For example:

If your current directory is...	...and you enter the command...	...your current directory becomes:
`/www`	`chdir figures`	`/www/figures`
`/www/figures`	`chdir ..`	`/www`
`/www/figures`	`chdir ../html`	`/www/html`
`/www/figures`	`chdir /html`	`/html`

7.4 Creating directories

Use the following to create a new directory:



Where:

directory
Name of the directory you want to create. This name can not contain wild cards.

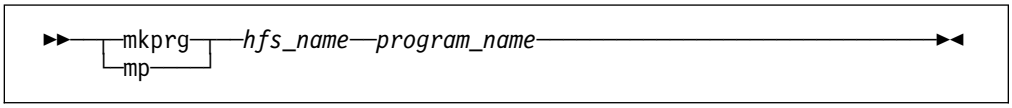
For example:

If your current directory is...	...and you enter the command...	...you make the new directory:
/www	mkdir figures	/www/figures
/www/figures	mkdir ../html	/www/html
/www/figures	mkdir /html	/html

7.5 Creating and changing programs

The ALCS hierarchical file system (HFS) allows you to create a special type of directory entry that associates an HFS name with an ECB-controlled program.

Use the following to create or change a directory entry for an ECB-controlled program:



Where:

hfs_name
HFS name you want to associate with the ECB-controlled program. This name can not contain wild cards.

program_name
Name (four characters) of the ECB-controlled program.

If *hfs_name* does not exist, the command creates the special directory entry to associate *hfs_name* with *program_name*.

If *hfs_name* is already associated with an ECB-controlled program, the command changes the special directory entry to associate *hfs_name* with the new *program_name*.

For example, if the name `res.cgi` does not exist in the current directory then the command:

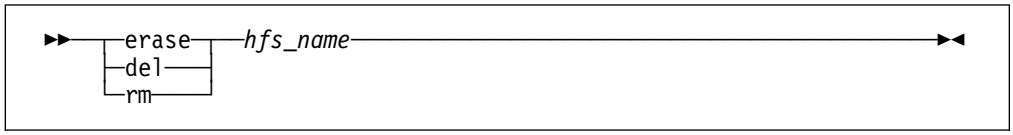
```
mkprg res.cgi www2
```

creates a special directory entry that associates `res.cgi` with the ECB-controlled program WWW2. If the name `res.cgi` is already associated with the ECB-controlled program WWW1, then the command changes the entry to associate `res.cgi` with WWW2.

Note that you can associate several different HFS names with the same ECB-controlled program.

7.6 Deleting directories, files, and programs

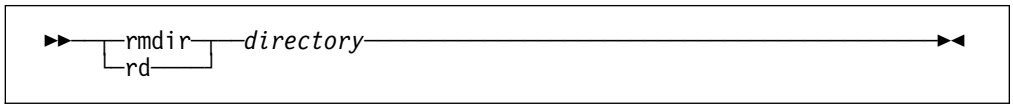
Use the following to delete a file or a directory entry for an ECB-controlled program:



Where:

hfs_name
HFS name of the file or ECB-controlled program directory entry you want to delete. This name can not contain wild cards.

Use the following to delete a directory:



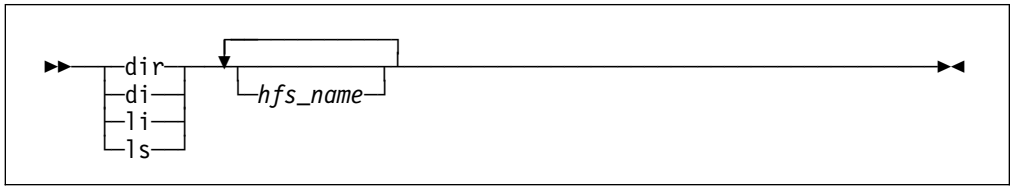
Where:

directory
Name of the directory you want to delete. This name can not contain wild cards.

Note that ALCS does not allow you to delete a directory unless the directory is empty.

7.7 Listing directories, files, and programs

Use the following to list directories, files, and programs:



Where:

hfs_name
HFS name of a directory, file, or program. This name can contain wild cards.

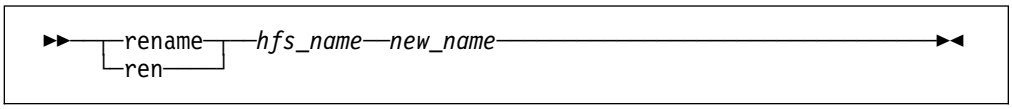
Note

Many hierarchical file systems recognize a directory name and list the *contents* of that directory. For example, if `www` is a directory, the OS/2 command `dir www` lists the files in the `www` directory.

ALCS does not currently do this. Of course, you can use `dir www/*` to list the files in the `www` directory.

7.8 Renaming directories, files, and programs

Use the following to rename directories, files, and programs:



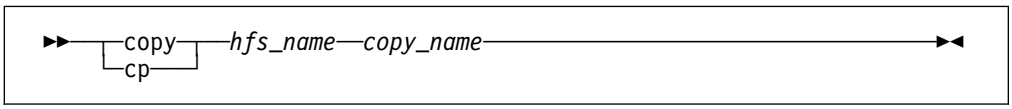
Where:

hfs_name
HFS name of a directory, file, or program. This name can not contain wild cards.

new_name
New name for the directory, file, or program. This name can not contain wild cards.

7.9 Copying files and programs

Use the following to copy a file or directory entry for an ECB-controlled program:



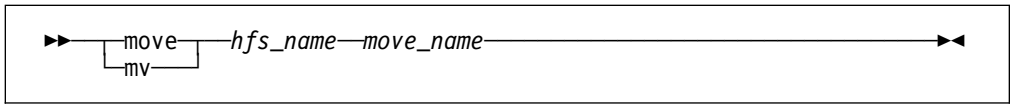
Where:

hfs_name
HFS name of the file or ECB-controlled program directory entry you want to copy. This name can not contain wild cards.

copy_name
Name of the copied file or program. This name can not contain wild cards.

7.10 Moving directories, files, and programs

Use the following to move a directory, file, or program:



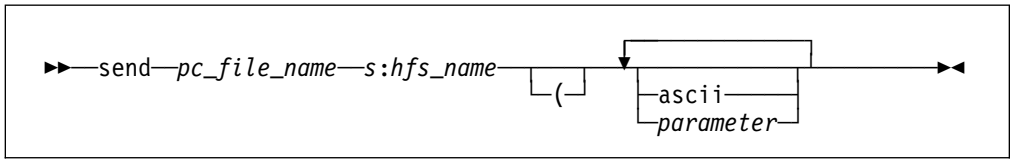
Where:

hfs_name
HFS name of the directory, file, or program that you want to move. This name can not contain wild cards.

move_name
Name of the moved directory, file, or program. This name can not contain wild cards.

7.11 Up-loading files from a PC

Use the following **in a PC DOS or OS/2 session** to up-load a file from a PC to the ALCS HFS:



Where:

pc_file_name
Name of the file on the PC. PC file transfer up-loads this file to the ALCS HFS.

s Host session name.

hfs_name
Name of the file in the ALCS HFS. This name can not contain wild cards.

ascii
The file is a text file, containing ASCII characters. If this parameter is omitted, the file is a binary file, for example, an image in GIF format.

Note that ALCS PC file transfer does *not* translate ASCII files to EBCDIC.

parameter
ALCS ignores any parameter other than **ascii**.

For example, the command:

```
send c:\webstuff\index.htm a:index.htm ( ascii
```

transfers the file `index.htm` from the `webstuff` directory on the PC's C disk to the current directory of the terminal using session A and calls it `index.htm`. The file is an ASCII text file and ALCS stores it in ASCII in the HFS.

OS/2 work-station with Communication Manager/2

If you are working at an OS/2 work-station with Communication Manager/2 (CM/2), you can use the “Transfer” pull-down menu in an ALCS CM/2 session to transfer files to the ALCS HFS:

1. From the “Transfer” pull-down menu, select “Send file to host...”.
2. Select the PC files you want to up-load.
3. If necessary, edit the entries to correct the names you want to use on the ALCS HFS.
4. Send the files.

It does not matter which you select from TSO, VM/CMS, or CICS – ALCS “understands” all three formats.

Be sure to select TEXT for ASCII text files, and BINARY for other files. Alternatively, set up your own type definitions (for example, ALCSASC for ASCII files, and ALCSBIN for binary files). To do this, in your 3270 emulation session:

1. From the “Transfer” pull-down menu, select “Setup”.
2. Select “Transfer type definitions...”.
3. Define your new transfer type.

You may find it useful to add transfer templates so that (for example) all PC files with a file-name extension of `htm` automatically have the same name on the ALCS HFS and have TEXT selected. To do this, in your 3270 emulation session:

1. From the “Transfer” pull-down menu, select “Setup”.
2. Select “Transfer templates...”.
3. Define your new transfer template.

Chapter 8. HTTP application interfaces

This chapter describes the entry conditions that ALCS Web server programs set up when they call your programs and the entry conditions that your programs must set up when they call ALCS Web server programs.

It also describes the fields in the HFS state block that you can use.

8.1 DSECT macros

8.1.1 DXCHFSS – HFS state block format

Use the DXCHFSS macro to generate a dummy section (DSECT) for the ALCS HFS state block. The format of the macroinstruction is:

DXCHFSS [REG=*reg***]**

Where:

REG=*reg*
Base register for the DSECT.

The DXCHFSS DSECT defines the following symbols that you can use:

Symbol	How you can use the symbol
DXCHFSS	Name of the DSECT. You can use this in USING instructions.
HS_WFITEM	You use this HFS directory entry to pass information about your response file to the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99). You should clear the whole directory entry to zeros before you fill in any subfields, using an instruction like: `XC HS_WFITEM,HS_WFITEM` To set up the field, you use the DXCHFSD field names (see 8.1.2, “DXCHFSD – HFS directory and data formats” on page 90).
HS_WRET	You use this field to pass the HTTP response code to the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99 and Appendix D, “HTTP response codes” on page 121).
HS_WAMET	You use this field to pass the HTTP methods that you allow to the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99). The field contains nine consecutive bytes, allowing you to pass up to nine allowed methods. You can use the following symbols to set (MVI) the methods:

Symbol	Method
HS_WMETG	GET
HS_WMETH	HEAD

HS_WMETP POST

Note: You *must* set the methods you allow when you set the "Method Not Allowed" (405) HTTP response code. If you do not allow any methods then you should set a different HTTP response code, for example, "Forbidden" (403).

HS_WOPG You use this field in your Web input editor (see 8.2, "Entry conditions from the Web server receiver" on page 94) to save the name of the ECB-controlled program that formats the HTTP response (your Web output editor). The ALCS Web server output interceptor calls this program (see 8.6, "Entry conditions from the Web server output interceptor" on page 99).

HS_WUSR Your Web programs can use this 256-byte field any way you want.

You can use the following symbols to refer to information about the original request. The ALCS Web server receiver initializes these fields before it calls your Web input editor (see 8.2, "Entry conditions from the Web server receiver" on page 94).

HS_WMET HTTP method, one byte. You can use the following symbols to test (CLI) the method:

Symbol	Method
HS_WMETG	GET
HS_WMETH	HEAD
HS_WMETP	POST

HS_WHTTV HTTP version major part in binary, one byte. For example, for HTTP 1.0, this field contains 1.

HS_WHTTR HTTP version minor part in binary, one byte. For example, for HTTP 1.0, this field contains 0.

HS_WRIP Originator IP address, four bytes.

Note that some firewalls (see 3.4, "Firewalls" on page 21) transform IP addresses. The ALCS Web server receiver stores the IP address that it receives. This may be different from the originator's real IP address.

HS_WRCV Greenwich mean time (GMT) date and time when the request arrived, eight bytes. The format of this field (all values in binary) is:

Bytes	Contents
0-1	Year (1 = 0001, Common Era)
2	Month in year (1 = January)
3	Day in month (1 = 1st)
4	Hour in day (0 = midnight)
5	Minute in hour
6	Second in minute
7	Reserved (zero).

For example, 18:45:22 on 4th July 1996 is stored as:

```
1996   7   4  18  45  22   0
```

HS_WIMS GMT "If-Modified-Since" date and time, eight bytes, or binary zeros if the request does not include an "If-Modified-Since" value. The format is the same as HS_WRCV.

HS_WREQ File address of the original request line, four bytes (see “Accessing the original request line – HS_WREQ” on page 89).

HS_WPTH File address of the “effective file name”, four bytes (see “Accessing the effective file name – HS_WPTH”).

HS_WAUTH File address of the “Authorization” value, four bytes, or binary zeros if the request does not include an “Authorization” value (see “Accessing the Authorization value – HS_WAUTH”).

HS_WUAG File address of the “User-Agent” value, four bytes, or binary zeros if the request does not include a “User-Agent” value (see “Accessing the User-Agent value – HS_WUAG” on page 90).

HS_WRCPL RCPL associated with the original input.

The ALCS RC0PL macro generates a DSECT for the RCPL.

Accessing the original request line – HS_WREQ

The field HS_WREQ in the HFS state block contains the file address of a short-term pool record (see “Short-term pool record format” on page 90) that contains the original request line.

Typical request lines are:

```
GET / HTTP/1.0
POST /cgi/ipars.cgi HTTP/1.0
```

Accessing the effective file name – HS_WPTH

The field HS_WPTH in the HFS state block contains the file address of a short-term pool record (see “Short-term pool record format” on page 90) that contains the effective file name.

This is the file name that the Web server uses to locate the requested object (in this case, your Web input editor) in the hierarchical file system directory. That is, it is the file name from the original request URL *after* ALCS “decodes” escape sequences (see 6.3, “File names in HTTP URLs” on page 77) and *after* it adds the name of your welcome page (see “Welcome pages” on page 55). For example:

Original URL	Effective file name
`http://www.someplace.com/`	`index.html`
`http://www.someplace.com/cgi/ipars.cgi`	`cgi/ipars.cgi`
`http://www.someplace.com/weird%3Bname.html`	`weird;name.html`

Accessing the Authorization value – HS_WAUTH

The field HS_WAUTH in the HFS state block contains the file address of a short-term pool record (see “Short-term pool record format” on page 90) that contains the Authorization value. If the request does not include an Authorization value, HS_WAUTH contains binary zeros.

For example, if you are using the basic authentication scheme, Authorization values comprise the word `'Basic'` followed by the user-ID and password, Base64 Content-Transfer-Encoded (see *MIME Part One*, RFC 1521). In this case, an Authorization value might be:

```
Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

(The weird looking string that starts `QWxhZG` is the Base64 Content-Transfer-Encoded representation of `Aladdin:open sesame`).

Accessing the User-Agent value – HS_WUAG

The field HS_WUAG in the HFS state block contains the file address of a short-term pool record (see "Short-term pool record format") that contains the User-Agent value. If the request does not include a User-Agent value, HS_WAUG contains binary zeros.

Typical User-Agent values include:

```
CERN-LineMode/3.0  libwww/3.0
Charlotte/1.2b3 VM_ESA/1.2.2 CMS/11
IBM WebExplorer DLL /v1.03a
Microsoft Internet Explorer/4.40.474beta (Windows 95)
Mozilla/2.0b5 (Macintosh; I; 68K)
NCSA_Mosaic/2.6 (X11;AIX 2 000001714100)  libwww/2.12 modified
```

Short-term pool record format

The fields HS_WREQ, HS_WPTH, HS_WAUTH, and HS_WUAG in the HFS state block can each contain the file address of a short-term pool record. The DXCHFSD DSECT (see 8.1.2, "DXCHFSD – HFS directory and data formats") defines the format of these short-term pool records.

The record ID is the value of the symbol `#RIDHFDT`. Use the macroinstruction:

```
DXCRID ACTION=KPTORD
```

to generate a definition for the symbol `#RIDHFDT`.

The data in the short-term pool record starts at field HF_PDATA. It is a string of EBCDIC characters, terminated with a null (`X'00'`) character.

8.1.2 DXCHFSD – HFS directory and data formats

Use the DXCHFSD macro to generate a dummy section (DSECT) for the directory entry (HS_WFITEM) in the ALCS HFS state block (see 8.1.1, "DXCHFSS – HFS state block format" on page 87), and for the output data blocks that you pass to the ALCS Web server sender (see 8.7, "Entry conditions to the Web server sender – CWW3" on page 99). The format of the macroinstruction is:

DXCHFSD [REG=*reg***]**

Where:

REG=*reg*
Base register for the DSECT.

The DXCHFSD DSECT defines the following symbols that you can use:

Symbol	How you can use the symbol
DXCHFSD	Name of the DSECT. You can use this in USING instructions.

Use the following symbols for the format of the HFS directory entry:

HF_FENT Start of the HFS directory entry. You can use this as an alternative to DXCHFSD in USING instructions.

HF_FNAM File name, or in the case of long names, the first 23 characters of the file name. If the file name is less than 23 characters, it must be followed by a null (X'00') character.

You use this field to pass a “working” file name of your output to the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99). This should not be the file name of any real file in the ALCS HFS.

The Web sender uses this name to determine the content type of the output (see 5.3.6, “Name qualifiers and metainformation” on page 57, 9.1, “Overriding content-type defaults” on page 103, and Appendix B, “File name qualifiers and HTTP metainformation” on page 115).

HF_FLEN Length in bytes of the file. You use this fullword field to pass the length in bytes of your output to the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99).

HF_FCOD Code of the file. You use this 1-byte field to pass the code of your output to the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99). You can use the following symbols to set (MVI) the code:

Symbol	Code
HF_FCODB	Binary
HF_FCODE	EBCDIC
HF_FCODA	ASCII

Use the following symbols for the format of the output data blocks that you pass to the ALCS Web server sender:

HF_PHEAD Standard record header. You can clear the whole record header to zeros using an instruction like:

```
XC    HF_PHEAD,HF_PHEAD
```

To set up the fields within the header, you use the following field names:

Symbol	How you can use the symbol
HF_PBID	Record ID. Set this to the symbol `#RIDHFDT`. Use the macroinstruction: `DXCRID ACTION=KPTORD` to generate a definition for the symbol `#RIDHFDT`.
HF_PCHK	Record code check. Set this to binary zeros.
HF_PCTL	Control byte. Set this to binary zeros.
HF_PFCH	Forward chain field. Set this to file address of the next record (L3 short-term pool file) in the chain, or to binary zeros if this is the only record.

HF_PDATA First byte of output data.

HF_PDATA_MAX The value of this symbol is the maximum number of bytes of data in any one data record. If the total size of your output data exceeds this value, you must use multiple blocks (file chained).

Each block except the last must contain exactly HF_PDATA_MAX bytes.

Programming notes

It is likely that your Web output editor refers to both the HFS directory entry in the HFS state block, and to the ouput data block(s).

You may find it useful to exploit labelled USING instructions to do this. For example:

```
         DXCHFSS ,                format of HFS state block
         DXCHFSD ,                format of HFS directory and data
         DXCRID ACTION=KPTORD     generate record ID symbols

⋮

         DXCHFSS REG=R02          use R02 for HFS state
DIR      USING HF_FENT,HS_WFITEM  use directory entry lables
DATA     USING DXCHFSD,R04        use R04 for data block(s)

         GETCC LEVEL=D4,SIZE=L3   get block for output data
         L     R04,CE1CR4         load base for output data

         XC    DATA.HF_PHEAD,DATA.HF_PHEAD clear data header
         MVC   DATA.HF_PBID,=AL2(#RIDHFDT) set up record ID

⋮
         L     R02,CE1CR3         load HFS state base

         MVC   HS_WRET,=C'200'    set normal response code
         MVI   DIR.HF_PCOD,HF_PCODE set code is EBCDIC
```

8.1.3 DXCWWPM – parsed HTTP input block format

Use the DXCWWPM macro to generate a dummy section (DSECT) for the parsed HTTP input block that the ALCS Web server receiver passes to your Web input editor (see 8.2, “Entry conditions from the Web server receiver” on page 94). The format of the macroinstruction is:

DXCWWPM [REG=*reg***]**

Where:

REG=*reg*
Base register for the DSECT.

The DXCWWPM DSECT defines the following symbols that you can use:

Symbol	How you can use the symbol
DXCWWPM	Name of the DSECT. You can use this in USING instructions.

All the following symbols refer to fullword fields.

WW_PDTA	The storage address (within the IMSG format input block) of the start of the request.

WW_PDTL Length (bytes) of the request.

WW_PBODY The storage address (within the IMSG format input block) of the start of the body in the request, or binary zeros if there is no body in the request.

WW_PBODL Length (bytes) of the body.

WW_PRURI The storage address (within the IMSG format input block) of the start of the URL in the request.

WW_PRURL Length (bytes) of the URL.

WW_RPATH The storage address (within the IMSG format input block) of the start of the path component of the URL. In ALCS, the path component of the URL is the file name. It starts with a `'/'` character. For example, the path `/myprog.cgi` corresponds to the HFS name `/www/myprog.cgi`

WW_RPATL Length (bytes) of the path.

WW_RPARM The storage address (within the IMSG format input block) of the parameters component (if any) in the URL, or binary zeros if there is no parameters component. The parameters component of the URL starts with a `';'` character.

WW_RPARL Length (bytes) of the parameters.

WW_RSRCH The storage address (within the IMSG format input block) of the query component (if any) in the URL, or binary zeros if there is no query component. The query component of the URL starts with a `'?'` character.

WW_RSRCL Length (bytes) of the query.

WW_RFRAG The storage address (within the IMSG format input block) of the fragment component (if any) in the URL, or binary zeros if there is no fragment component. The fragment component of the URL starts with a `'#'` character.

WW_RSRCL Length (bytes) of the fragment.

8.1.4 DXCWWPF – parsed HTTP forms data block format

Use the DXCWWPF macro to generate a dummy section (DSECT) for the parsed HTTP forms data block that the ALCS Web server receiver passes to your Web input editor (see 8.2, “Entry conditions from the Web server receiver” on page 94). The format of the macroinstruction is:

DXCWWPF [REG=*reg*]

Where:

REG=*reg*
Base register for the DSECT.

The DXCWWPF DSECT defines the following symbols that you can use:

Symbol	How you can use the symbol
DXCWWPF	Name of the DSECT. You can use this in USING instructions.
WW_PRNUM	Halfword containing the number of keyword/value pairs in the request. The maximum value is 60.
WW_PRENT	First keyword/value pair entry.

Each keyword/value pair entry contains the following fields that you can use.

WW_PRKLN	One byte containing the length (characters) of the keyword, the maximum length is 8 characters.
WW_PRKWD	The keyword.
WW_PRVLN	One byte containing the length (characters) of the value, the maximum length is 52 characters.
WW_PRVAL	The value.

Programming notes

You can increment the DSECT base register by the length of WW_PRENT to process the entries in turn, like this:

```
         DXCWWPF REG=R02           use R02 for parsed message block
         L     R02,CE1CR2          load parsed message block base
         LH    R03,WW_PRNUM        load number of entries
         SR    R01,R01             clear for inserts

LOOP     DC    0H'0'
         IC    R01,WW_PRKLN        insert length of keyword

⋮
         LA    R02,L'WW_PRENT(,R02) step to next entry
         BCT   R03,LOOP            and loop through entries
```

8.2 Entry conditions from the Web server receiver

The ALCS Web server receiver looks-up the file name in the request URL. If the HFS directory entry associates the name with an ECB-controlled program (a Web input editor) (see 7.5, “Creating and changing programs” on page 81) then the receiver calls that program.

On entry to the Web input editor, there are storage blocks attached at the following ECB storage levels:

ECB level	Contents
D0	The input request from the Web browser in IMSG format. Before the Web server receiver passes this to your input editor, it makes the following changes: • It translates the input from ASCII to EBCDIC. • It translates HTTP header field names (but not the field values) into upper case. • It unfolds folded HTTP header fields (if you do not know what this means, you do not need to worry about it).

D1 The parsed HTTP input block (see 8.1.3, “DXCWWPM – parsed HTTP input block format” on page 92).

D2 The parsed forms data block (see 8.1.4, “DXCWWPF – parsed HTTP forms data block format” on page 93), or free if there is no forms data in the request.

D3 The HFS state block (see 8.1.1, “DXCHFSS – HFS state block format” on page 87). ECB data level D3 is initialized with the correct record ID, record code check, and file address. Note that the HFS state block file address is held. You must unhold it (FILUC or UNFRC) when you no longer need the HFS state block.

Other ECB levels are free.

Programming notes

You can have Web input editors that handle any type of request from a Web browser. All you need to do is define an HFS directory entry that associates a file name in the HFS with the name of a Web input editor. The ALCS Web server receiver will then pass all requests for that name to your program.

A common use for Web input editors is to process requests from forms that your ALCS Web server sends to the browser. In this case, the HTML for your form specifies the file name that the request will contain, the names of the input fields (these are “keywords”) and the values that the input fields can take.

If you want to use the parsed forms data block on level D2, you should restrict the form to define a maximum of 60 input fields, each field name to a maximum of at most 8 characters, and each value to a maximum of at most 52 characters. If you do not respect these limits, you can still process forms input, but you must parse the body of the request yourself (the body is the part of the request that contains the forms input).

Be aware that the ALCS Web server receiver does *not* translate forms input data to upper case (it does translate it to EBCDIC though).

Also be aware that your Web input editor can not assume that a request which specifies the corresponding HFS name necessarily comes from a properly completed form. This means your Web input editor must (at least) check:

- The request method is as expected, for example:

```
         CLI   HS_WMET,HS_WMETP    is the request method POST
         BNE   ERROR1              no - send Method Not Allowed (405)

  ⋮
ERROR1   DC    0H'0'
         MVI   HS_WAMET,HS_WMETP   indicate POST method allowed
         MVC   HS_WRET,=C'405'     set response code
         XC    HS_WFITEM,HS_WFITEM clear directory entry
         RELCC D2,TYPE=COND        release parsed forms data (if any)
         RELCC D4,TYPE=COND        release HTTP input data (if any)
         RELCC D5,TYPE=COND        release parsed HTTP data (if any)
         ENTDC CWW3                and send error response
```

- The forms input exists, for example:

```
         LEVTA D2,                 check forms data is attached
               NOTUSED=ERROR2      no - send Input Error (400)

⋮
ERROR2   DC    0H'0'
         MVC   HS_WRET,=C'400'     set response code
         XC    HS_WFITEM,HS_WFITEM clear directory entry
         RELCC D4,TYPE=COND        release HTTP input data (if any)
         RELCC D5,TYPE=COND        release parsed HTTP data (if any)
         ENTDC CWW3                and send error response
```

- The forms input contains the expected keywords and values.

8.3 Dynamically selecting Web pages from programs

You can dynamically select a Web page from a Web program. To do this you must issue a BACKC macro with storage blocks attached at the following ECB levels.

ECB level	Contents
D0	Optional. See level D0 from the Web server receiver.
D1	Optional. See level D1 from the Web server receiver.
D2	Optional. See level D2 from the Web server receiver.
D3	The HFS state block (see 8.1.1, “DXCHFSS – HFS state block format” on page 87). ECB data level D3 must be initialized with the correct record ID, record code check and file address. The HFS state block address must be held.
DF	Storage block containing the path of the requested dynamic page. The page may exist in any directory of the HFS. If a relative path is used, the Web server root directory `/www` is assumed. The path must refer to a valid file or program object currently on the HFS.

Programming notes

The program that gains control is the Web server receiver – CWW1.

You may select file or program objects. If you select a program object, then this is known as nesting. ALCS allows up to 5 levels of program nesting. If you attempt to select a program beyond the 5th nesting level, then an error will occur. ALCS assumes you have entered an unrecoverable loop.

You might dynamically select standard response messages from a “`/messages`” directory like this:

```
         DXCWWPF REG=R02           FORMS DATA DSECT
         L     R02,CE1CR2          LOAD FORMS BLOCK
         .
         .
         LA    R01,WW_PRVAL        ADDRESS VALUE

         LA    R03,MSG001          ADDRESS MSG001 PATH
         LA    R04,L'MSG001        AND LOAD LENGTH (FOR EXECUTE)
         CLC   =C'A',0(R01)        VALUE = A
         BE    LABEL1              YES - GO SERVE MESSAGE

         LA    R03,MSG002          ADDRESS MSG002 PATH
         LA    R04,L'MSG002        AND LOAD LENGTH (FOR EXECUTE)
         CLC   =C'B',0(R01)        VALUE = B
         BE    LABEL1              YES - GO SERVE MESSAGE
         B     ERROR               OTHERWISE - ERROR
         .
         .
LABEL1   EQU   *
         GETCC LEVEL=DF,SIZE=L1    GET BLOCK FOR DYNAMIC PATH ON DF
         L     R06,CE1CRF          LOAD STORAGE BLOCK ADDRESS

         MVC   0(0,R06),0(R03)     EXECUTED INSTRUCTION
         EX    R04,*-6             MOVE IN NEW DYNAMIC PATH

         BACKC ,                   AND GO SEND THE DYNAMIC PAGE
         .
         .
MSG001   DC    C'/messages/msg001.html',AL1(0)
MSG002   DC    C'/messages/msg002.html',AL1(0)
MSG003   DC    C'/messages/msg003.html',AL1(0)
```

Since "`/messages`" is not a sub-directory of the Web server root directory "`/www`", then the Web pages it contains cannot be served directly. This is an advantage as you would not want people just browsing `msg001.html` for example as it is both irrelevant to them and more importantly, could contain sensitive information.

8.4 Server Side Include (SSI)

The ALCS Web server will parse output files for server side include commands if the content type of the file is `text/x-ssi-html`. This defaults to files with the extension `.shtml`. An SSI command is an HTML extract of the following form:

`<!--#directive tag1=`*`value1`* `tag2=`*`value2`* `-->`

Values may be enclosed in double quotes.

#include directive

The `#include` directive inserts text of another file into the parsed file, replacing the SSI command. The format is:

`<!--#include file=`*`included_file`* `-->`

The included file must be specified as an absolute filename. It must not be a program or a directory. If the included file cannot be found then the SSI command is removed and no text is inserted. Parsing then continues.

You may include files that also require parsing. This is known as nesting. The ALCS Web server allows up to 5 levels of nesting. If you attempt to include beyond the fifth nesting level, an error will occur.

8.5 Creating an entry for the Web server output interceptor

If you want to exploit the ALCS Web server output interceptor program, you must find the places where your application issues SENDC or ROUTC to send responses. In each case, you replace the SENDC macro with a sequence of instructions that issues a CREMC instead of the SENDC when the response is to the Web.

If the originating CRI is in EBROUT, and the OMSG block is on level D6, you might use a sequence like this:

```
         COMIC CRI=EBROUT,         retrieve resource information      X
               DATA=SYS,           for the originating CRI            X
               AREA=(0,ICELEN)
         CLI   ICEATY,TPTCPIP      is response to the Web
         BNE   LABEL1              no - branch to SENDC

         CREMC CWW2,LEVEL=D6,      pass the OMSG to CWW2              X
               DATA=0,LENGTH=0
         B     LABEL2              bypass the SEND

LABEL1   DC    0H'0'
         SENDC D6,A                send the response

LABEL2   DC    0H'0'
```

If there are two OMSG blocks, on levels D6 and D5, you might use a sequence like this:

```
         COMIC CRI=EBROUT,         retrieve resource information      X
               DATA=SYS,           for the originating CRI            X
               AREA=(0,ICELEN)
         CLI   ICEATY,TPTCPIP      is response to the Web
         BNE   LABEL1              no - branch to SENDC

         FLIPC D6,D5               flip OMSGs to correct order
         CREMC CWW2,LEVEL=(D5,D6), pass the OMSGs to CWW2             X
               DATA=0,LENGTH=0
         FLIPC D5,D6               restore OMSGs to previous levels
         B     LABEL2              bypass the SEND

LABEL1   DC    0H'0'
         SENDC (D6,D5),M           send the response

LABEL2   DC    0H'0'
```

Alternatively, your application may simply DISPC SEND the response to the Web. ALCS will convert the response into OMSG format that the output interceptor program CWW2 expects. ALCS then issues the CREMC to CWW2. ALCS returns to the application that issued the DISPC SEND with the DISPC built blocks released.

8.6 Entry conditions from the Web server output interceptor

If you exploit the ALCS Web server output interceptor program (see 8.5, “Creating an entry for the Web server output interceptor” on page 98) then the interceptor retrieves the HFS state block and checks the contents of field HS_WOPG.

If HS_WOPG contains binary zeros then the interceptor sends the output message directly to the Web browser as unformatted text (content-type `text/plain`). We do not expect that this is satisfactory for production use, but it may be useful during debugging.

If HS_WOPG contains the name of an ECB-controlled program (a Web output editor) then the interceptor enters (ENTDC) that program.

On entry to the Web output editor, there are storage blocks attached at the following ECB storage levels:

ECB level	Contents
D0	The first OMSG block.
D1	The second OMSG block, or free if there is only one OMSG block.
D3	The HFS state block (see 8.1.1, “DXCHFSS – HFS state block format” on page 87). ECB data level D3 is initialized with the correct record ID, record code check, and file address. Note that the HFS state block file address is held.

Other ECB levels are free.

The RCPL area of the ECB (CE1RCPL) contains the RCPL associated with the original input.

Register	Contents
R15=0	There is only one output message block, or the second output message block on level D1 was not truncated.
R15=4	The second output message block on level D1 was truncated.

8.7 Entry conditions to the Web server sender – CWW3

You call (ENTDC) the ALCS Web server sender, CWW3, to build a header for your response (if applicable) and to send the response to the Web browser.

You must set the following entry conditions for CWW3:

ECB level	Contents
D3	The HFS state block (see 8.1.1, “DXCHFSS – HFS state block format” on page 87). ECB data level D3 must be initialized with the correct record ID, record code check, and file address. The HFS state block file address must be held. You must initialize the following fields in the HFS state block before you call CWW3:

HS_WFITEM Clear the whole directory entry to zeros before you fill in any subfields, using an instruction like:

```
XC    HS_WFITEM,HS_WFITEM
```

Then set up the following fields within HS_WFITEM:

HF_FNAM The "working" file name
HF_FLEN The total length (bytes) of your response
HF_FCOD The code of your response (probably EBCDIC, HF_FCODE)

HS_WRET The HTTP response code (see Appendix D, "HTTP response codes" on page 121).

HS_WAMET The HTTP methods that you allow. You must set the methods you allow when you set the "Method Not Allowed" (405) HTTP response code.

D5 An L3 storage block containing the first or only part of your response. The block must be in DXCHFSD format (see 8.1.2, "DXCHFSD – HFS directory and data formats" on page 90).

If the output requires more than one block, the forward chain field (HF_PFCH) of the first block (on D5) must contain the file address of an L3 short-term pool record which contains more of the response. This second block is also in DXCHFSD format and can contain the file address of a third block, and so on. Each block except the last must contain exactly `HF_PDATA_MAX` bytes.

All other ECB levels must be free.

The RCPL area of the ECB (CE1RCPL) must contain the RCPL associated with the original input. If you use the Web server output interceptor, you do not need to do anyhing about this; the Web server output interceptor initializes CE1RCPL as required.

CWW3 does not require any particular values in general registers or in the ECB work areas.

8.8 Entry conditions to the Web server sender – CWW5

You can also call (ENTDC) another Web server sender program, CWW5. This can be used when an HTML response is sent to the Web browser.

Before calling CWW5, you must build your HTML response lines by using the DISPC ADD macro. One DISPC ADD call can add one, or many lines of HTML to your response.

Delimit each line of HTML by either the #CR character followed by the #LF character, or by simply the #CAR character.

When using a DISPC ADD macro to add some HTML to your response, you must include at least one #CAR character per 240 HTML characters added. If not, the HTML lines may be broken at unexpected places.

You must set the following entry conditions for CWW5:

ECB level	Contents
D3	The HFS state block (see 8.1.1, “DXCHFSS – HFS state block format” on page 87). ECB data level D3 must be initialized with the correct record ID, record code check, and file address. The HFS state block file address must be held. You must initialize the following fields in the HFS state block before calling CWW5: **HS_WFITEM** Clear the whole directory entry to zeros before you fill in any subfields, using an instruction like: `XC HS_WFITEM,HS_WFITEM` Then set up the following fields within HS_WFITEM: **HF_FNAM** The “working” file name. **HS_WRET** The HTTP response code (see Appendix D, “HTTP response codes” on page 121). **HS_WAMET** The HTTP methods that you allow. You must set the methods you allow when you set the “Method Not Allowed” (405) HTTP response code.

All other ECB levels must be free.

The RCPL area of the ECB (CE1RCPL) must contain the RCPL associated with the original input. If you use the Web server output interceptor, you do not need to do anything about this; the Web server output interceptor initializes CE1RCPL as required.

Register	Contents
14	Level on which the HTML response was built using DISPC ADD macros (not level D3 or D5).
15	Zero.

CWW5 does not require any particular values in any other of the general registers (except general registers 14 (RDA) and 15 (RDB)), or in the ECB work areas.

Example:

```
         DISPC ADD,TEXT=HTML1,LENGTH=LEN1   ADD BLOCK OF HTML
         DISPC ADD,TEXT=HTML2,LENGTH=LEN2   ADD BLOCK OF HTML

         LA    R14,D2                       LOAD DISPC BLOCK LEVEL

         ENTDC CWW5                         CONVERT AND SEND HTML

         .
         .

HTML1    DC    C'<html><head>'
         DC    AL1(#CR,#LF)
         DC    C'<title>HTML Example</title>'
         DC    C'</head><body>'
         DC    AL1(#CR,#LF)
         DC    C'<p>This is an example of some HTML.'
LEN1     EQU   *-HTML1

HTML2    DC    C'<p>We use DISPC ADD macros followed by a '
         DC    C'call to CWW5 to send the HTML. '
         DC    C'Each line of HTML may be terminated by #CR and '
         DC    C'#LF or by just #CAR.'
         DC    AL1(#CAR)
         DC    C'</body></html>'
LEN2     EQU   *-HTML2
```

8.9 Reading the HFS state block

You can read the HFS state block by calling the ECB-controlled program CHFS. On entry to CHFS, set general register 14 (RDA) to the level where you want to read the HFS state block; this must not be level D0 and there must not be a storage block already attached. Set general register 15 (RDB) to the originating CRI. For example:

```
LA    R14,D3              request read on level D3
SR    R15,R15             clear for insert
ICM   R15,B'0111',EBROUT  insert originator CRI
ENTRC CHFS                read the HFS state
```

On return, the HFS state block is attached at the specified storage level. The associated data level is initialized with the record ID, RCC, and file address. The HFS state block file address is held.

All registers (except R14 and R15) are returned unchanged. All ECB areas (except the specified storage and data level) are returned unchanged.

Chapter 9. Installation-wide exits for the Web

This chapter describes the installation-wide exits that you can use to customize the ALCS Web server.

9.1 Overriding content-type defaults

By default, ALCS determines the content-type of a file that it sends as an HTTP response by examining the final qualifier of the file name. For example, it assumes that the content-type of the file `myfile.htm` is `text/html`.

B.1, “File name qualifiers for content type” on page 115 lists the qualifiers that ALCS recognizes and their associated content types.

ALCS provides two ECB-controlled installation-wide exits that you can use to change the way ALCS determines the content-type of a file.

Note

ALCS generates some HTTP responses with file names that have `html` and `txt` as the final qualifier. If you use an unusual naming convention for your files, be sure to set the content-type for these to `text/html` and `text/plain` respectively.

Also, some Web browsers make assumptions about the content-type of a file based on the file name. If you use a an unusual naming convention for your files, these browsers may not present the files correctly. For example, if you have a JPEG image file called `picture.html`, some browsers attempt to interpret the file as HTML (which obviously does not work).

9.1.1 AWM1 – custom content-type method

The ALCS Web sender calls program AWM1 (if it exists) with the following entry conditions:

R01 Address of the complete file name, delimited with a null (`X'00'`).

R02 Address of the last qualifier in the file name.

R03 Length (bytes) of the last qualifier in the file name. R03 contains binary zeros if the file name comprises a single qualifier.

R06 Binary zeros.

For example, if the file name is `example.thing.html`:

R01 Address of `example.thing.html`
R02 Address of `html`
R03 Length of `html` (4)
R06 Binary zeros.

AWM1 must return (BACKC) to the calling program, with the following return conditions:

For example, if the file name is `example.thing.html`:

R01 Unmodified
R02 Unmodified
R03 Unmodified
R06 Binary zeros or address of content-type.

If AWM1 returns a nonzero value in R06, it must be the address of a one-byte binary length field (which does not include itself), immediately followed by the character-string content-type. For example, it might return the address of the following assembler language constant:

```
        DC    AL1(13),C'text/richtext'
```

If AWM1 returns binary zeros in R06, ALCS determines the content-type by examining the final qualifier of the file name. Note that this includes using the installation-wide exit table in AWM2 (if it exists).

AWM1 can corrupt the contents of general registers R00, R07, R14, and R15. It must not corrupt any other registers or ECB areas.

9.1.2 AWM2 – custom content-type table

The ALCS Web sender uses the program AWM2 (if it exists) as a table that associates file-name qualifiers with content-types. AWM2 must be a table of entries; each entry contains:

1. A one-byte binary length field (which includes itself). This is the length in bytes of the entire entry.
2. A one-byte binary length field (which does not include itself). This is the length in bytes of the qualifier.
3. A character string which is the qualifier.
4. A one-byte binary length field (which does not include itself). This is the length in bytes of the content-type.
5. A character string which is the content-type.

For example, a single entry might be:

```
TYPE001  DC    AL1(TYPE002-TYPE001)   length of complete entry
         DC    AL1(4),C'html'         length and qualifier
         DC    AL1(9),C'text/html'    length and content-type
TYPE002  DC    ...                    next entry in table
```

You can (optionally) include an entry for file names that have only a single qualifier (for example, `myfile`). This entry specifies a length of zero for the qualifier (it omits the qualifier). For example:

```
TYPE050  DC    AL1(TYPE051-TYPE050)   length of complete entry
         DC    AL1(0)                 no qualifier
         DC    AL1(10),C'text/plain'  length and content-type
TYPE051  DC    ...                    next entry in table
```

You must include a special entry at the end of your table. This entry specifies a length of zero for the entire entry and a length of zero for the qualifier (it omits the qualifier). It can (optionally) include a default content-type which ALCS uses if there is no previous matching entry in the table. For example:

```
TYPELAST DC    AL1(0)                 indicates last entry
         DC    AL1(0)                 no qualifier
         DC    AL1(11),C'www/unknown' length and default content-type
```

If you do not include a default content-type, and if there is no matching entry in your table, then ALCS determines the content-type as shown in B.1, “File name qualifiers for content type” on page 115. In this case, your last entry type must comprise three bytes of binary zeros. For example:

```
TYPELAST DC    AL1(0)                 indicates last entry
         DC    AL1(0)                 no qualifier
         DC    AL1(0)                 no default content-type
```

Chapter 10. Messages and codes

10.1 Messages

DXC6000I **OK**

Explanation: The ALCS HFS command completed normally.

DXC6010E **Unknown command**

Explanation: The input command is not supported by the ALCS hierarchical file system (HFS) application.

User Response: Correct the typing or syntax error and retry the command.

DXC6011E **Incorrect command format**

Explanation: The input command format is not correct.

User Response: Correct the typing or syntax error and retry the command.

DXC6012E **Error processing command**

Explanation: The ALCS hierarchical file system (HFS) application was unable to process the command. For example:

- You attempted to delete or rename a file that does not exist.
- You attempted to rename a file to a the name of another existing file.
- The name you specified on a `cd` or `rd` command is not the name of a directory.

User Response: Retry the command using a valid HFS file name or directory name.

DXC6013E **Unauthorized for this command**

Explanation: You attempted to use the ALCS hierarchical file system (HFS) but your user ID is not authorized to issue commands which update the HFS.

User Response: If you believe you should be authorized to update the HFS, then contact the person in your organization who has responsibility for allocating user IDs and so on. This person may be called your security coordinator, your system programmer, or may have some other title.

DXC6020E **Current directory invalid – reset to root**

Explanation: The name recorded as your current directory is not valid. Another user may have deleted the directory.

System Action: Your current directory is reset to the root.

10.2 System error numbers

000360 **CWW1 – UNKNOWN ORIGINATOR**

Explanation: The ALCS Web server received input with an unknown originator CRI.

User Response: This error should not occur. If it does, contact your IBM programming representative.

000361 **CWW1 – ORIGINATOR NOT TCP/IP**

Explanation: The ALCS Web server received input with an originator CRI that is not defined as a TCP/IP resource.

User Response: This error should not occur. If it does, contact your IBM programming representative.

000362 **CWW1 – HFS UNUSABLE**

Explanation: The ALCS Web server was not able to access the ALCS hierarchical file system (HFS).

Operator Response: Notify your system programmer. The ALCS Web server is unusable.

System Programmer Response: You may be able to recover the ALCS HFS by restoring the record #KPTRI ordinal 9 from the ALCS data base update log.

Alternatively, you can reinitialize the ALCS HFS by resetting the record ID of #KPTRI(9) to binary zeros, using the `ZAFIL` command. This deletes the entire contents of the ALCS HFS – you must reload all your files from your PC copies.

Alternatively, restore the ALCS real-time data base.

000363 **CWW3 – UNABLE TO READ HFS OBJECT**

Explanation: The ALCS Web server was not able to access an object in the ALCS hierarchical file system (HFS).

Operator Response: Notify your system programmer.

System Programmer Response: Reload the object from your PC copy.

000364 **CWW5 – INVALID ECB LEVEL SPECIFIED**

Explanation: A program entered ECB-controlled program CWW5, but the data level specified in general register 14 (RDA) is invalid.

System Action: The ALCS Web server sends an error response.

Programmer Response: Correct the programming error.

000365 **CWW5 – STORAGE LEVEL NOT IN USE**

Explanation: A program entered ECB-controlled program CWW5, but the level specified in general register 14 (RDA) is not in use.

System Action: The ALCS Web server sends an error response.

Programmer Response: Correct the programming error.

000366 **CWW5 – INVALID RECORD ID**

Explanation: A program entered ECB-controlled program CWW5, but the storage block attached at the specified ECB level does not contain a record belonging to the ALCS output file (created by DISPC ADD macro).

System Action: The ALCS Web server sends an error response.

Programmer Response: Correct the programming error.

000367 **CWW5 – NO LINES IN OUTPUT FILE**

Explanation: A program entered ECB-controlled program CWW5, but no DISPC ADD calls were issued to add one or more HTML text lines to the ALCS output file.

System Action: The ALCS Web server sends an error response.

Programmer Response: Correct the programming error.

000368 **CWW5 – CANNOT RETRIEVE OUTPUT MSG RECORD**

Explanation: A find error occurred on the retrieval of an ALCS output message record while building a Web server sender data record.

System Action: The ALCS Web server sends an error response.

System Programmer Response: Examine the system error dump and determine the cause of the error.

- If it is due to a record ID error, check whether the database is corrupted (for example, because of pool problems).
- If it is due to a hardware error, get the unit serviced.

000369 **CWW1 – INCORRECT RETURN CONDITIONS FROM WEB PROGRAM**

Explanation: A Web program returned to program CWW1 using BACKC in order to select a Web page, but the return conditions were incorrect.

System Action: The ALCS Web server sends an error response.

Programmer Response: Correct the Web program to ensure the correct conditions are set up before you return to CWW1 using BACKC.

00036A **CWW1 – LOOP DETECTED IN DYNAMIC PAGE SELECTION**

Explanation: A Web program returned to program CWW1 using BACKC in order to select a Web page, but the maximum number of nested Web programs selected has been reached. ALCS assumes an unrecoverable loop has occurred.

System Action: The ALCS Web server sends an error response.

System Programmer Response: You may only dynamically select up to a maximum of 5 nested Web programs. Check your Web programs to ensure that no more than 5 nested Web programs are selected. Check also that your Web program is not selecting itself.

00036B **CWW6 – UNABLE TO READ HFS OBJECT**

Explanation: The ALCS Web server was not able to access an object in the ALCS hierarchical file system (HFS).

Operator Response: Notify your system programmer.

System Programmer Response: Reload the object from your PC copy.

00036C **CWW6 – HFS UNUSABLE**

Explanation: The ALCS Web server was not able to access the ALCS hierarchical file system (HFS).

Operator Response: Notify your system programmer. The ALCS Web server is unusable.

System Programmer Response: You may be able to recover the ALCS HFS by restoring the record #KPTRI ordinal 9 from the ALCS data base update log.

Alternatively, you can reinitialize the ALCS HFS by resetting the record ID of #KPTRI(9) to binary zeros, using the `ZAFIL` command. This deletes the entire contents of the ALCS HFS – you must reload all your files from your PC copies.

Alternatively, restore the ALCS real-time data base.

00036D **CWW6 – LOOP DETECTED IN SSI PROCESSING**

Explanation: Web pages may include the text of another file by using the `#include` SSI directive. This file may also include the text of another file. This is known as nesting. ALCS has detected more than 5 levels of nesting. ALCS assumes an unrecoverable loop has occurred

System Action: The ALCS Web server sends an error response.

System Programmer Response: Correct your Web pages. Ensure that no more than 5 levels of `#include` nesting is present in your pages. Check also that your pages do not `#include` themselves.

00036E **UNABLE TO READ HFS STATE CONTROL BLOCK**

Explanation: ALCS could not read the hierarchical file system (HFS) state block for a resource.

System Action: ALCS allocates a new HFS state block and continues.

000370 **UNABLE TO READ HFS DIRECTORY**

Explanation: ALCS was not able to access a directory in the ALCS hierarchical file system (HFS).

000371 **LOGIC ERROR – WILDC WORK AREA TOO SMALL**

Explanation: An internal error occurred in the ALCS hierarchical file system (HFS) processing. The work area provided to WILDC was too small.

System Action: ALCS terminates the entry.

System Programmer Response: If this error occurs contact your IBM programming representative.

000373 **UNABLE TO OPEN HFS**

Explanation: ALCS was not able to open the ALCS hierarchical file system (HFS).

System Programmer Response: You may be able to recover the ALCS HFS by restoring the record #KPTRI ordinal 9 from the ALCS data base update log.

Alternatively, you can reinitialize the ALCS HFS by resetting the record ID of #KPTRI(9) to binary zeros, using the `ZAFIL` command. This deletes the entire contents of the ALCS HFS – you must reload all your files from your PC copies.

Alternatively, restore the ALCS real-time data base.

000374 **UNABLE TO READ HFS LONG-NAME**

Explanation: ALCS was not able to access the component of the ALCS hierarchical file system (HFS) that contains the full name of an object with a long name.

System Action: ALCS terminates the entry.

System Programmer Response: Inform your IBM programming support representative.

00037A **FILE TRANSFER – UNEXPECTED INPUT**

Explanation: The terminal (PC) which initiated the file transfer transmitted data to ALCS which does not conform to the IBM 3270-PC file transfer protocol.

System Action: ALCS terminates the file transfer.

User Response: Inform your PC administrator. If this PC is able to transfer files successfully to other IBM host systems such as VM/CMS or MVS/TSO, then ask your system programmer to inform your IBM programming support representative.

00037B **FILE TRANSFER – NOT DDN TERMINAL**

Explanation: The terminal (PC) which initiated the file transfer does not support the 3270 Distributed Data Management (DDN) protocol.

System Action: ALCS terminates the file transfer.

User Response: Inform your PC administrator. If this PC is able to transfer files successfully to other IBM host systems such as VM/CMS or MVS/TSO, then ask your system programmer to inform your IBM programming support representative.

Appendix A. ALCS generation parameters for the Web server

You must run some ALCS generations to configure ALCS to run the Web server. The following sections describe these.

A.1 SCTGEN – including TCP/IP support

You must include TCP/IP support in your ALCS configuration if you plan to use the ALCS Web server. Use the following SCTGEN parameter to do this:

```
[label] SCTGEN TCPIP={YES|(YES,threads)}
              ,...
```

Where:

TCPIP={YES|(YES,*threads*)}
Include TCP/IP support in your ALCS configuration. You may want to specify a nondefault number of threads (*threads*) if you have other TCP/IP applications.

A.2 COMGEN – specifying VTAM receive buffer size

ALCS's PC file-transfer support uses VTAM receive buffers. You should check the file-transfer buffer size that your PC uses and either adjust it to match your ALCS VTAM receive buffer size, or increase the size of your ALCS VTAM receive buffers.

Microsoft Windows work-station with IBM Personal Communications

If you are working at a Microsoft Windows work-station with IBM Personal Communications, you adjust the file-transfer buffer size that your PC uses as follows:

1. Select the Personal Communications “Edit” menu
2. Select “Preferences”
3. Select “Transfer”
4. On the “General” tab, ensure the “Packet Size” does not exceed ALCS's VTAM receive buffer size.

ALCS uses a default VTAM receive buffer size that is large enough to contain an input message for a size L3 storage block (normally 4000 bytes). If your installation defines a larger storage block size, you can use the COMGEN macroinstruction to increase ALCS's VTAM receive buffer size as follows:

```
COMGEN ...,                                                    X
      RCVSZE=MAX
```

A.3 COMDEF – defining the Web server and HFS applications

You must define the ALCS Web server and the HFS application as ALCS applications. Use COMDEF macroinstructions to do this, as follows:

```
COMDEF UPDATE=ADD,LDTYPE=ALCSAPPL,                              X
      NAME=WWW1,PROG=CWW1,FMSG=YES
COMDEF UPDATE=ADD,LDTYPE=ALCSAPPL,                              X
      NAME=HFS1,PROG=CHFP,FMSG=NO
```

You must specify the program names CWW1 for the Web server and CHFP for the HFS application. Of course, you do not need to use the names `WWW1` and `HFS1`, you can use any valid ALCS application names that you chose.

A.4 COMDEF – defining the Web server communication resource

You must define the ALCS Web server communication resource. Use a COMDEF macroinstruction to do this. The COMDEF macroinstruction format that you use is:

```
COMDFLT LDTYPE=TCPIP
COMDEF NAME=WWW80,TERM=(SERVER,HTTP),APPL=WWW1,LPORT=80,      X
      MAXCONN=100
```

A.5 SEQGEN – defining the Web server access log sequential file

You must define the WWW real-time sequential file. The ALCS Web server writes the access log to this sequential file. Use an SEQGEN macroinstruction to do this, as follows:

```
SEQGEN NAME=WWW,                                                X
      TYPE=RT,                                                  X
      UNIT=(3380,1),                                            X
      DISP=(NEW,CATLG,CATLG),                                   X
      DSNAME=XAN.WWW.SEQF.WEBLOG,                               X
      LABEL=(,,,OUT,RETPD=0),                                   X
      RECFM=VB,                                                 X
      VOLCNT=1,                                                 X
      BUFNO=2,                                                  X
      BLKSIZE=13000,                                            X
      SPACE=(1000,100),                                         X
      LRECL=12900
```

You must specify `NAME=WWW` and `TYPE=RT`. You can select other parameters to suit your installation.

Appendix B. File name qualifiers and HTTP metainformation

This appendix lists the file name qualifiers that the ALCS Web server recognizes and uses to build metainformation in the HTTP header of Web responses.

B.1 File name qualifiers for content type

The ALCS Web server recognizes the following file name qualifiers as content type indicators:

Qualifier	Content type	Description
Text types		
htm	text/html	HTML text
html	text/html	HTML text
htmls	text/html	HTML text
java	text/plain	Java source
shtml	text/x-ssi-html	HTML text with server-side includes
txt	text/plain	Plain ASCII text
Image types		
bmp	image/bmp	BMP format image
gif	image/gif	GIF format image
ief	image/ief	
jpg	image/jpeg	JPEG format image
jpe	image/jpeg	JPEG format image
jpeg	image/jpeg	JPEG format image
tif	image/tiff	TIFF format image
tiff	image/tiff	TIFF format image
Audio types		
au	audio/basic	UNIX audio clip
snd	audio/basic	
Video types		
avi	video/x-msvideo	AVI format video clip
mov	video/quicktime	QuickTime format video clip
mpg	video/mpeg	MPEG format video clip
mpe	video/mpeg	MPEG format video clip
mpeg	video/mpeg	MPEG format video clip
qt	video/quicktime	QuickTime format video clip
Application types		
123	application/vnd.lotus-1-2-3	Lotus 1-2-3 spreadsheet
ai	application/postscript	PostScript output
apr	application/vnd.lotus-approach	Lotus Approach document
bin	application/octet-stream	Unformatted binary data
class	application/octet-stream	Java applet
eps	application/postscript	PostScript output
lwp	application/vnd.lotus-wordpro	Lotus Wordpro document
oda	application/oda	
or2	application/vnd.lotus-organizer	Lotus Organiser document
or3	application/vnd.lotus-organizer	Lotus Organiser document
org	application/vnd.lotus-organizer	Lotus Organiser document
pdf	application/pdf	Adobe portable document format
pre	application/vnd.lotus-freelance	Lotus Freelance document
prz	application/vnd.lotus-freelance	Lotus Freelance document

Qualifier	Content type	Description
ps	application/postscript	PostScript output
sam	application/vnd.lotus-wordpro	Lotus Wordpro document
scm	application/vnd.lotus-screencam	Lotus ScreenCam document
vew	application/vnd.lotus-approach	Lotus Approach document
wk1	application/vnd.lotus-1-2-3	Lotus 1-2-3 spreadsheet
wk3	application/vnd.lotus-1-2-3	Lotus 1-2-3 spreadsheet
wk4	application/vnd.lotus-1-2-3	Lotus 1-2-3 spreadsheet

Other types

mime	www/mime

If the file name comprises a single qualifier (for example, `myfile`) then ALCS uses the default content type `text/plain`. If the file name comprises more than one qualifier but ALCS does not recognize the final qualifier (for example, `myfile.mytype`) then ALCS uses the default content type `www/unknown`.

You can use an installation-wide exit to change these content-type associations, including the defaults, see 9.1, “Overriding content-type defaults” on page 103.

Appendix C. HTTP uniform resource locators

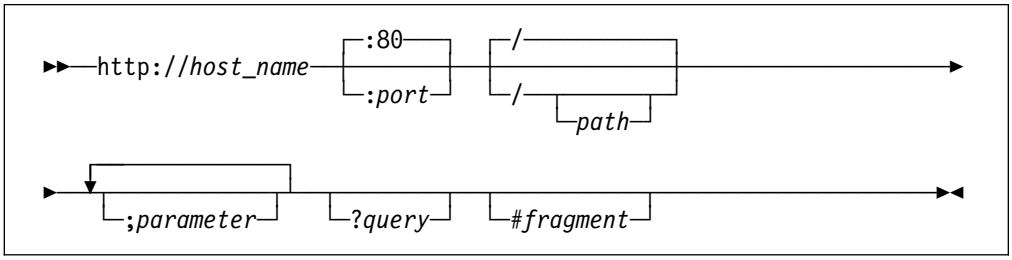
A uniform resource locator (URL) is a string of characters that uniquely identifies a resource on the Internet. If you have a private intranet, you can also use URLs to identify resources on your intranet. URLs can identify a variety of different types of resource, called **schemes**. For example, a URL can identify:

a destination address for electronic mail
a file located on a file transfer protocol (FTP) server
a resource (file or program) on an HTTP server
and so on.

In this section, we only describe the URL for HTTP (World Wide Web) server resources.

C.1 HTTP URL format

The format of an HTTP (Web) URL is:



Where:

host_name
Fully-qualified host name for the Web server. The host name is case-insensitive.

port
Port where the Web server listens, up to 4 decimal digits.

path
Uniquely identifies the object to the Web server. The interpretation of *path*, including whether it is case-sensitive or case-insensitive, depends on the Web server implementation, or on Web server configuration options, or both.

In this book, we call *path* the file name.

For the ALCS Web server, *path* is case-sensitive. It is the hierarchical file system (HFS) name of a file or a program. The HFS name is relative to the ALCS server root.

parameter
Parameter passed to the object.

If *path* is the HFS name of a file, the ALCS Web server ignores any *parameter*s. If it is the HFS name of a program, the program can use *parameter*s any way it wants.

query
Search argument or similar passed to the object.

If *path* is the HFS name of a file, the ALCS Web server ignores any *query*s. If it is the HFS name of a program, the program can use *query*s any way it wants.

fragment
Label that identifies a particular location within an object.

Web browsers use the *fragment* component of the URL to scroll automatically to a specified location within a page.

If *path* is the HFS name of a file, the ALCS Web server ignores any *fragment*. If it is the HFS name of a program, the program should also ignore any *fragment*.

C.2 Characters allowed in HTTP URLs

HTTP URLs must not contain the following characters:

Control (ASCII X'00' through X'1F')
Space (ASCII X'20')
Double-quote (ASCII X'22')
Percent (ASCII X'25'), except in escape sequences
Less-than (ASCII X'3C')
Greater-than (ASCII X'3E')
DEL (X'7F')

The *path* and *parameter* components of HTTP URLs must not contain the following characters:

Hash or pound (ASCII X'23')
Semi-colon (ASCII X'3B')
Query (ASCII X'3F')

The *query* and *fragment* components of HTTP URLs must not contain the following character:

Hash or pound (ASCII X'23')

The ALCS hierarchical file system (HFS) allows file names to include characters that are not allowed in URLs. We recommend that you do not use these characters in ALCS HFS names, but if you do, your users must use URL escape sequences for them.

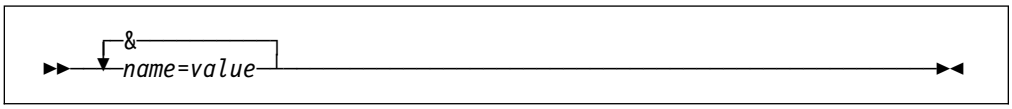
A URL escape sequence comprises a percent (%) followed by the hexadecimal code for the ASCII character.

For example, to request the file `weird;name.html`, you use `weird%3Bname.html` or `weird%3bname.html` in the URL (X'3B' is the ASCII code for the semicolon). To request the file `100%.html`, you use `100%25.html` in the URL (X'25' is the ASCII code for the percent).

Note: The ALCS Web server does not allow the *path* component of a URL to contain the escape sequence `%2F` or `%2f`.

C.3 HTTP URL parameters and queries

The URL definition does not specify the format of parameters and queries, and the ALCS Web server does not require any specific format. However there is a convention that you can chose to use for the format. The convention is that a parameter or query comprises one or more name/value pairs, separated by ampersand (&) characters, as follows:



You use a plus (+) characters to represent spaces in *value*. For example, a URL might include a date and time as a query, like this:

```
http://someplace.com/someprog.cgi?Date=15+Sep&Time=14:40
```

Note: The ALCS Web server does *not* interpret any URL escape sequences in parameters or queries.

Appendix D. HTTP response codes

HTTP defines a number of response codes. The following list is not intended to be a complete list of all HTTP response codes. It includes only those that you might want to specify for the ALCS Web server sender (see 8.7, “Entry conditions to the Web server sender – CWW3” on page 99). If you want more detail, you should refer to other documentation, such as Internet Requests For Comments (RFCs).

The following codes indicate successful completion of the request.

200 OK.

This is the default response code unless the length of the response is zero.

202 Accepted.

You can use this code when your application generates the response before it completes processing the request. The response should indicate when you expect processing to complete and how the user can check that it completes OK.

204 No Content.

This is the default response code if the length of the response is zero.

The following codes indicate that you are not sending a response, but that the Web browser can take some action to get a response. You should set the length of your response to zero for these codes.

304 Not Modified.

The request included an “if-modified-since” date and time. The requested information is older than that date and time.

The following codes indicate that the request is incorrect. You should include an explanation of the problem in your response for these codes.

400 Bad Request.

The syntax of the request is incorrect.

403 Forbidden.

Your application does not allow access to the requested information or service.

404 Not Found.

Your applications should not use this code. ALCS uses it when the request specifies a file name that is not defined in the ALCS HFS.

405 Method Not Allowed.

Your application does not allow access to the requested information or service using the method specified in the request.

You must set the methods you *do* allow when you use this code. If you do not, then ALCS resets your response code to 403.

408 Request Time-out.

Your applications should not use this code.

410 Gone.

The requested information or service is no longer available.

You might want to use this code tell Web browsers that you have withdrawn a service; for example, that you no longer allow airline seat reservations from the Web. It is *not* appropriate for information or services that you have moved to a different Web server.

414 Request-URI Too Large.

Your applications should not use this code. ALCS uses it when the request URL is too long for ALCS to process.

The following codes indicate that there is an error or restriction in your application that prevents you from processing the request. You should include an explanation of the problem in your response for these codes.

500 Internal Server Error.

Your application encountered an error condition that prevents you from processing the request. You might want to use this code if your application is unable to read a record from the data base, or if it finds the data is corrupted.

501 Not Implemented.

Your applications should not use this code. ALCS uses it when the request specifies a method that ALCS does not implement.

503 Service Unavailable.

Your application is temporarily unable to process the request. You might want to use this code if your application uses the `LODIC` macro to check how busy ALCS is, and wants to reject a request because ALCS is too busy.

List of Abbreviations

AAA agent assembly area
ASCII american national standard code for information interchange
CGI common gateway interface (also, in airline applications: carrier gross index)
CR carriage return
CRI communication resource identifier
CRLF carriage return, line feed
DES Data Encryption Standard
DNS domain name server
EBCDIC extended binary coded decimal interchange code
FAQ frequently asked questions
FTP file transfer protocol
GDS global distribution system
GIF graphic interchange format
GMT Greenwich mean time
HTH host-to-host
HTML hypertext markup language
HTTP hypertext transmission protocol
IAB Internet Architecture Board
IANA Internet Assigned Numbers Authority
ICANN The Internet Corporation for Assigned Names and Numbers
ID Internet Draft
IETF Internet Engineering Task Force
InterNIC Internet Network Information Center
IP Internet protocol
IPARS international programmed airline reservation system
IRTF Internet Research Task Force
ISOC Internet Society
JPEG Joint Photographic Experts Group
LF line feed
MIME multipurpose internet mail extensions
MPEG Moving Pictures Expert Group
NAT network address translation
NCSA National Computer Security Association
PERL practical extraction and reporting language
PNR passenger name record
POSIX portable operating system interface
PTF program temporary fix
RFC Internet Request for Comments
TCP transmission control protocol
TIFF tag image file format
TSO time sharing option
URI uniform resource identifier
URL uniform resource locator
W3C The World Wide Web Consortium
WWW World Wide Web

Bibliography

Web sites

We have found the following Web sites give useful information about the World Wide Web. But please note that:

- There are a vast number of interesting and useful sites on the Web. The following list is not intended to be comprehensive.
- The Web is a very dynamic place. New Web sites are created all the time and old ones often dissapear without trace. The following list contains URLs that are working sites at the time we are writing this.
- By listing a Web site here, we do not imply that IBM necessarily approves of or agrees with the content of the sites.

We follow a common convention of showing URLs enclosed by '<' and '>'. Omit these characters when you type a URL into your Web browser.

IBM and Lotus sites

<http://www.ibm.com/>
The IBM Corporation.

This site provides news and information about IBM products and services.

<http://www.lotus.com/>
The Lotus Development Corporation.

This site provides news and information about Lotus products and services.

Web search sites

These sites provide indexes to the World Wide Web (links to other Web sites) organized for easy access, or search facilities that help you locate Web pages that contain information you are interested in, or both.

<http://www.ibm.com/search/>
Searches IBM sites.

<http://www.google.com/>
Google

<http://www.yahoo.com/>
Yahoo!

<http://www.ask.com/>
Ask.com

<http://www.msn.com/>
MSN Search (replaced by Windows Live Search in September 2006)

<http://www.live.com/>
Windows Live Search

<http://www.directory.net/>
Open Directory Project's directory of commercial services, products, and information on the Internet.

In addition to these specialist search sites, many other Web sites provide search facilities.

Information on the Internet and the World Wide Web

<http://www.icann.org/>
The Internet Corporation for Assigned Names and Numbers (ICANN). ICANN iwas created in September 1998 in order to oversee a number of Internet-related tasks previously performed directly on behalf of the U.S. Government by other organizations, notably the Internet Assigned Numbers Authority (IANA). The tasks of ICANN include managing the assignment of domain names and IP addresses (before September 1998, this was managed by InterNIC or Internet Network Information Center).

<http://www.isoc.org/>
The Internet Society (ISOC). The Internet Society is a non-governmental International organization for global cooperation and coordination for the Internet and its internetworking technologies and applications. Its major standards efforts are directed through the IETF and IAB (see below).

<http://www.iab.org/>
The Internet Architecture Board (IAB). The IAB is the committee charged with oversight of the technical and engineering development of the Internet by the Internet Society (ISOC). It oversees a number of Task Forces, of which the most important are the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF).

<http://www.ietf.org/>
The Internet Engineering Task Force (IETF). The IETF is the protocol engineering, development, and standardization arm of the Internet Architecture Board (IAB), a technical advisory group of the Internet Society. The IETF is a large open international community of network designers, operators, vendors, and researchers concerned with the

evolution of the Internet protocol architecture and the smooth operation of the Internet.

<http://www.irtf.org/>
The Internet Research Task Force (IRTF). The IRTF is a sister group to the IETF. Its stated mission is It is composed of research groups that study long-term issues relating to the Internet and related technologies.

<http://www.w3.org/>
The World Wide Web Consortium (W3C). The W3C is an industry consortium which seeks to promote standards for the evolution of the Web and interoperability between WWW products by producing specifications and reference software. Although W3C is funded by industrial members, it is vendor-neutral, and its products are freely available to all.

<http://www.w3schools.com/html/>
W3Schools' HTML tutorial

Internet Drafts, Requests for Comments (RFCs), and standards

Internet **Drafts** are informal documents that give information about work in progress. You can get Internet Drafts from a number of sources, including the IETF at <http://www.ietf.net/>.

Some Internet Drafts evolve into Internet **Requests for Comments** (RFCs). RFCs are authoritative formal documents that contain a wealth of useful information. Technically, they specify Internet standards track protocols for the Internet community, and request discussion and suggestions for improvements.

You can get RFCs from a number of Web sites, including the IETF at <http://www.ietf.net/> and the RFC Editor at <http://www.rfc-editor.org/>.

Some RFCs become accepted as **Internet Standards**. You can find out about these at <http://www.rfc-editor.org/>.

The following list includes RFCs which we have found useful and which we think you might find useful too.

1521 *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*

1630 *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*

1738 *Uniform Resource Locators (URL)*

1866 *Hypertext Markup Language – 2.0*

1867 *Form-based File Upload in HTML*

1918 *Address Allocation for Private Internets*

1942 *HTML Tables*

1945 *Hypertext Transfer Protocol – HTTP/1.0*

1980 *A Proposed Extension to HTML: Client-Side Image Maps*

1983 *Internet Users' Glossary*

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z