



# Handling the Client Request: HTTP Request Headers

*Core Servlets & JSP book:* [www.coreservlets.com](http://www.coreservlets.com)  
*More Servlets & JSP book:* [www.moreservlets.com](http://www.moreservlets.com)  
*Servlet and JSP Training Courses:* [courses.coreservlets.com](http://courses.coreservlets.com)

Slides © Marty Hall, <http://www.coreservlets.com>, book © Sun Microsystems Press

1

## Agenda

- **Idea of HTTP request headers**
- **Reading request headers from servlets**
- **Example: printing all headers**
- **Common HTTP 1.1 request headers**
- **Example: compressing Web pages**
- **Example: password-protecting Web pages**

2

# Handling the Client Request: HTTP Request Headers

- **Example HTTP 1.1 Request**

```
GET /search?keywords=servlets+jsp HTTP/1.1
```

```
Accept: image/gif, image/jpg, */*
```

```
Accept-Encoding: gzip
```

```
Connection: Keep-Alive
```

```
Cookie: userID=id456578
```

```
Host: www.somebookstore.com
```

```
Referer: http://www.somebookstore.com/findbooks.html
```

```
User-Agent: Mozilla/4.7 [en] (Win98; U)
```

- **It shouldn't take a rocket scientist to realize that you need to understand HTTP to be effective with servlets or JSP**

# Reading Request Headers (Methods in HttpServletRequest)

- **General**

- `getHeader`
- `getHeaders` (2.2 only)
- `getHeaderNames`

- **Specialized**

- `getCookies`
- `getAuthType` and `getRemoteUser`
- `getContentLength`
- `getContentType`
- `getDateHeader`
- `getIntHeader`

- **Related info**

- `getMethod`, `getRequestURI`, `getProtocol`

# Checking For Missing Headers

- **HTTP 1.0**
  - *All* request headers are optional
- **HTTP 1.1**
  - Only `Host` is required
- **Conclusion**
  - *Always* check that `request.getHeader` is non-null before trying to use it

```
String val = request.getHeader("some name");
if (val != null) {
    ...
}
```

# Printing All Headers

```
public class ShowRequestHeaders extends HttpServlet {
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Servlet Example: Showing Request Headers";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=#FDF5E6>\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<B>Request Method: </B>" +
            request.getMethod() + "<BR>\n" +
            "<B>Request URI: </B>" +
            request.getRequestURI() + "<BR>\n" +
            "<B>Request Protocol: </B>" +
            request.getProtocol() + "<BR><BR>\n" +
```

# Printing All Headers (Continued)

```
        "<TABLE BORDER=1 ALIGN=CENTER>\n" +
        "<TR BGCOLOR=\"#FFAD00\">\n" +
        "<TH>Header Name<TH>Header Value");
Enumeration headerNames = request.getHeaderNames();
while(headerNames.hasMoreElements()) {
    String headerName = (String)headerNames.nextElement();
    out.println("<TR><TD>" + headerName);
    out.println("    <TD>" + request.getHeader(headerName));
}
out.println("</TABLE>\n</BODY></HTML>");
}

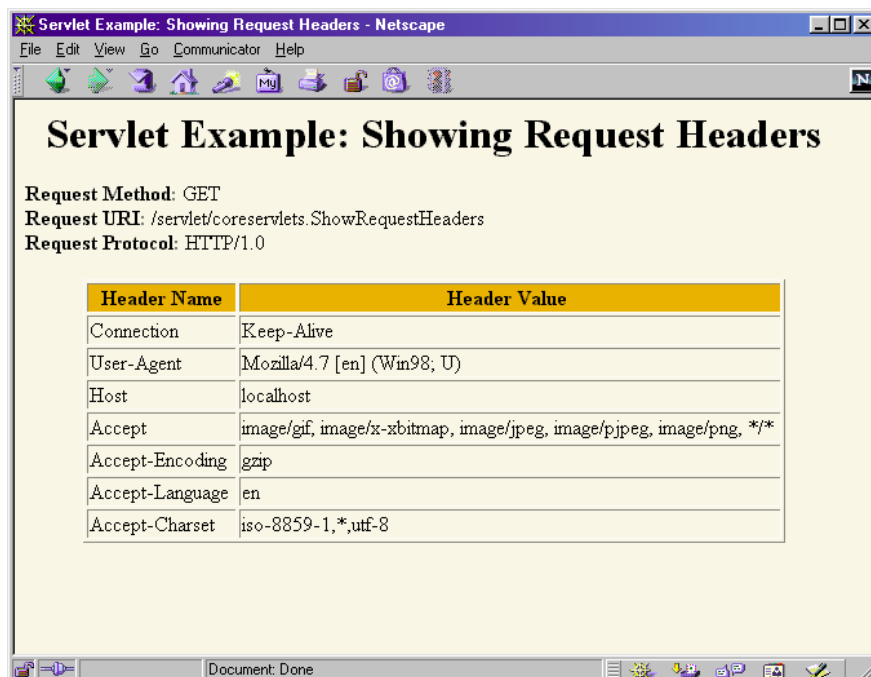
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}
}
```

7

Request Headers

www.coreservlets.com

# Printing All Headers: Typical Netscape Result



Servlet Example: Showing Request Headers - Netscape

File Edit View Go Communicator Help

Servlet Example: Showing Request Headers

Request Method: GET  
Request URI: /servlet/coreservlets.ShowRequestHeaders  
Request Protocol: HTTP/1.0

Header Name	Header Value
Connection	Keep-Alive
User-Agent	Mozilla/4.7 [en] (Win98; U)
Host	localhost
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding	gzip
Accept-Language	en
Accept-Charset	iso-8859-1,*,utf-8

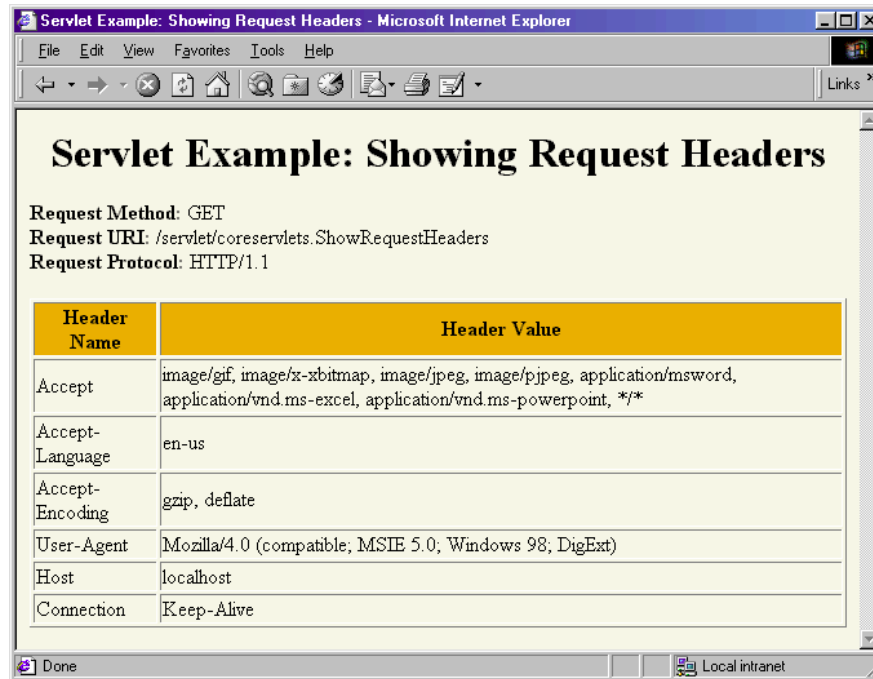
Document: Done

8

Request Headers

www.coreservlets.com

# Printing All Headers: Typical Internet Explorer Result



The screenshot shows a Microsoft Internet Explorer window titled "Servlet Example: Showing Request Headers". The address bar shows the URL "/servlet/coreservlets.ShowRequestHeaders". The page content displays the following information:

Request Method: GET  
Request URI: /servlet/coreservlets.ShowRequestHeaders  
Request Protocol: HTTP/1.1

Header Name	Header Value
Accept	image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/msword, application/vnd.ms-excel, application/vnd.ms-powerpoint, */*
Accept-Language	en-us
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
Host	localhost
Connection	Keep-Alive

9

Request Headers

www.coreservlets.com

## Common HTTP 1.1 Request Headers

- **Accept**
  - Indicates MIME types browser can handle
  - Can send different content to different clients. For example, PNG files have good compression characteristics but are not widely supported in browsers. A servlet could check to see if PNG is supported, sending `<IMG SRC="picture.png" ...>` if it is supported, and `<IMG SRC="picture.gif" ...>` if not.
  - Warning: IE incorrectly sets this header when you hit the Refresh button. It sets it correctly on original request.
- **Accept-Encoding**
  - Indicates encodings (e.g., gzip or compress) browser can handle.
  - See following example

10

Request Headers

www.coreservlets.com

## Common HTTP 1.1 Request Headers (Continued)

- **Authorization**

- User identification for password-protected pages.
- See upcoming example.
- Instead of HTTP authorization, use HTML forms to send username/password and store info in session object. This approach is usually preferable because standard HTTP authorization results in a small, terse dialog box that is unfamiliar to many users.
- Servers have high-level way to set up password-protected pages without explicit programming in the servlets.
  - For details, see Chapter 7 (Declarative Security) and Chapter 8 (Programmatic Security) of *More Servlets and JavaServer Pages*, [www.moreservlets.com](http://www.moreservlets.com).

## Common HTTP 1.1 Request Headers (Continued)

- **Connection**

- In HTTP 1.0, keep-alive means browser can handle persistent connection. In HTTP 1.1, persistent connection is default. Persistent connections mean that the server can reuse the same socket over again for requests very close together from the same client (e.g., the images associated with a page, or cells within a framed page).
- Servlets can't do this unilaterally; the best they can do is to give the server enough info to permit persistent connections. So, they should set Content-Length with `setContentLength` (using `ByteArrayOutputStream` to determine length of output). See example in book.

- **Cookie**

- Gives cookies previously sent to client. Use `getCookies`, not `getHeader`. See chapter & later class session.

## Common HTTP 1.1 Request Headers (Continued)

- **Host**

- Indicates host given in original URL
- This is a *required* header in HTTP 1.1. This fact is important to know if you write a custom HTTP client (e.g., WebClient used in *CSAJSP* Chapter 2) or telnet to a server and use the HTTP/1.1 version.

- **If-Modified-Since**

- Indicates client wants page only if it has been changed after specified date
- Don't handle this situation directly; implement `getLastModified` instead. See example in *CSAJSP* Chapter 2.

## Common HTTP 1.1 Request Headers (Continued)

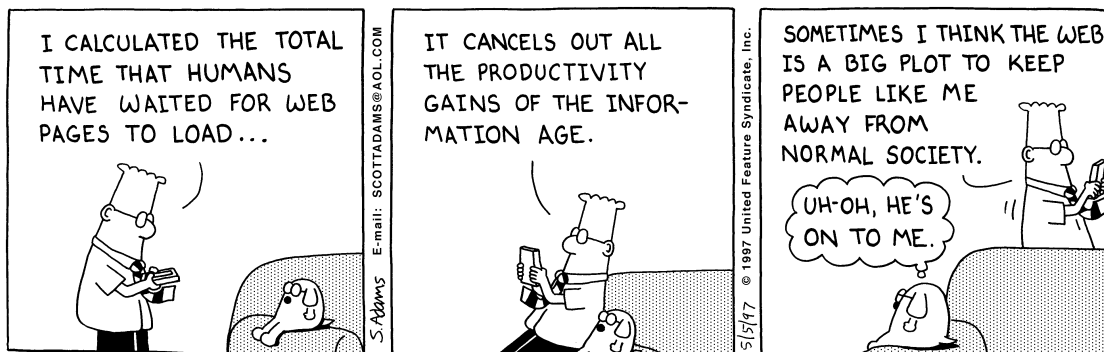
- **Referer**

- URL of referring Web page
- Useful for tracking traffic; logged by many servers
- Can also be used to let users set preferences and then return to the page they came from
- Can be easily spoofed, so don't let this header be your sole means of deciding (for example) how much to pay sites that show your banner ads.
- Some personal firewalls, Norton in particular, screen this out and replace it with `Weferer` (and random chars as values)

- **User-Agent**

- String identifying the browser making the request
- Best used for identifying *category* of client
  - Web browser vs. I-mode cell phone, etc.
- For Web applications, use other headers if possible
- Again, can be easily spoofed.

# Sending Compressed Web Pages



Dilbert used with permission of United Syndicates Inc.

# Sending Compressed Pages: EncodedPage.java

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    String encodings = request.getHeader("Accept-Encoding");
    String encodeFlag = request.getParameter("encoding");
    PrintWriter out;
    String title;
    if ((encodings != null) &&
        (encodings.indexOf("gzip") != -1) &&
        !"none".equals(encodeFlag)) {
        title = "Page Encoded with GZip";
        OutputStream out1 = response.getOutputStream();
        out = new PrintWriter(new GZIPOutputStream(out1), false);
        response.setHeader("Content-Encoding", "gzip");
    } else {
        title = "Unencoded Page";
        out = response.getWriter();
    }
}
```

# Sending Compressed Pages: EncodedPage.java (Continued)

```
out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n");
String line = "Blah, blah, blah, blah, blah. " +
            "Yadda, yadda, yadda, yadda.";
for(int i=0; i<10000; i++) {
    out.println(line);
}
out.println("</BODY></HTML>");
out.close();
}
```

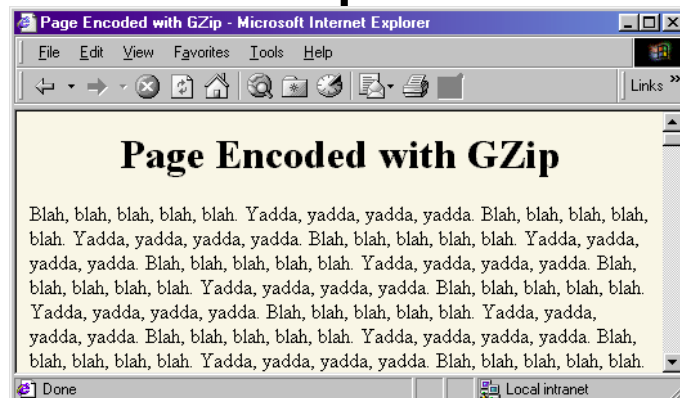
17

Request Headers

www.coreservlets.com

# Sending Compressed Pages: Results

- Uncompressed (28.8K modem), Netscape 4.7 and Internet Explorer 5.0: **> 50 seconds**
- Compressed (28.8K modem), Netscape 4.7 and Internet Explorer 5.0: **< 5 seconds**
- Caution: be careful about generalizing benchmarks



18

Request Headers

www.coreservlets.com

# Restricting Access to Web Pages

- **Main approach: "declarative" security via web.xml settings**
  - See *More Servlets and JSP* for lots of detail
- **Alternative: programmatic HTTP**
  - 1 Check whether there is Authorization header. If not, go to Step 2. If so, skip over word "basic" and reverse the base64 encoding of the remaining part. This results in a string of the form username:password. Check the username and password against some stored set. If it matches, return the page. If not, go to Step 2.
  - 2 Return a 401 (Unauthorized) response code and a header of the following form:  
WWW-Authenticate: BASIC realm="some-name"  
This instructs browser to pop up a dialog box telling the user to enter a name and password for some-name, then to reconnect with that username and password embedded in a single base64 string inside the Authorization header.

19

Request Headers

www.coreservlets.com

# SecretServlet (Registered Name of ProtectedPage Servlet)

```
public class ProtectedPage extends HttpServlet {
    private Properties passwords;
    private String passwordFile;

    public void init(ServletConfig config)
        throws ServletException {
        super.init(config);
        try {
            passwordFile =
                config.getInitParameter("passwordFile");
            passwords = new Properties();
            passwords.load(new FileInputStream(passwordFile));
        } catch(IOException ioe) {}
    }
}
```

20

Request Headers

www.coreservlets.com

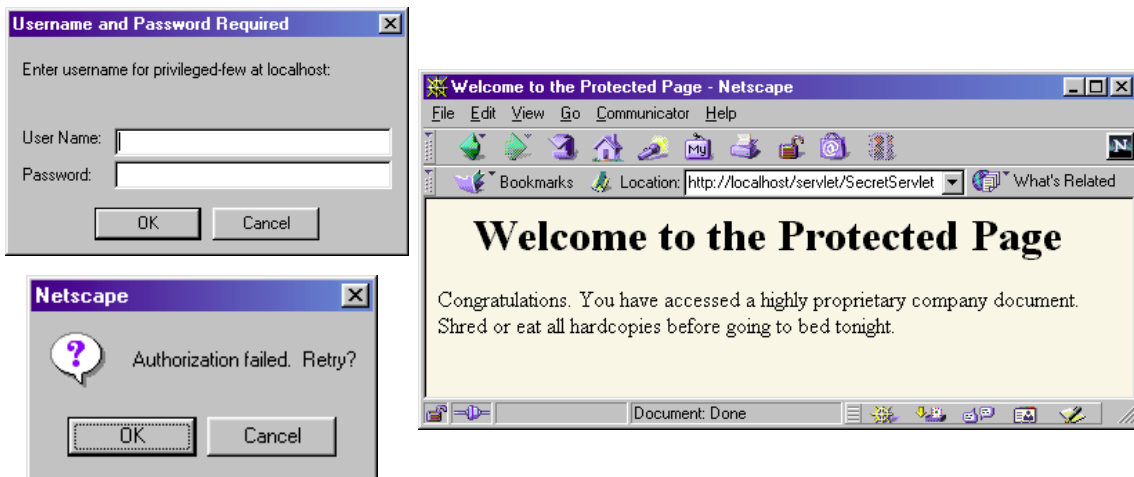
## SecretServlet (Continued)

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String authorization =
        request.getHeader("Authorization");
    if (authorization == null) {
        askForPassword(response);
    } else {
        String userInfo =
            authorization.substring(6).trim();
        BASE64Decoder decoder = new BASE64Decoder();
        String nameAndPassword =
            new String(decoder.decodeBuffer(userInfo));
        // Check name and password
    }
}
```

## SecretServlet (Continued)

```
private void askForPassword
    (HttpServletResponse response) {
    // SC_UNAUTHORIZED is 401
    response.setStatus(response.SC_UNAUTHORIZED);
    response.setHeader("WWW-Authenticate",
        "BASIC realm=\"privileged-few\"");
}
```

# SecretServlet In Action



## Summary

- Many servlet tasks can *only* be accomplished by making use of HTTP headers coming from the browser
- Use `request.getHeader` for arbitrary header
  - Remember to check for `null`
- Cookies, authorization info, content length, and content type have shortcut methods
- Most important headers you read directly
  - Accept
  - Accept-Encoding
  - Connection
  - Referer
  - User-Agent