Track Your Users With EAServer

A Sybase White Paper by Berndt Hamboeck

technology

WHITE PAPER

Table of Contents

Introduction

Maybe you do not want to treat all of your users of your Web site the same way. Not all of them are allowed to see the content on your Web server, or you simply want to know who uses what on YOUR machine. Or you want to give different users a different color scheme on your page. Well, to bring it down to a single point: you want to know what your users are doing on your Web server. We want to get you started down this road.

In this article, we'll look at how we can tailor content for a particular user. Of course, in order to tailor the content for a user we need to be able to identify the user. You should already know how we can keep track of users across sessions, and in this article, we will put those techniques to practical use.

What Can We Expect to Get?

As we mentioned it might be very useful to get an idea of how visitors use your Web site. What pages do they visit? In which order do they visit them? We can learn a great deal about a client by examining the request parameters and HTTP headers that they send when they access a site. We can then use this information to build a better picture of our users, and tailor the content of the site accordingly. However, we don't want to have to create the code to keep track of our users ourselves. It would be much better to use an existing application to do this tedious job for us, so we'll look at some software that performs this task.

Specifically, we'll look at:

- How we can gather information from a request by a client, and what sort of information is available
- How we can use visitor-tracking software to keep track of the pages that are accessed on our site

Snooping on Requests

It's often useful to know details about clients, and to get information about the requests that a Web application is handling. Java™ Servlets have a number of methods available to gain access to this information. This information can then be used for security purposes and for identifying the client. It is retrieved from the `ServletRequest` object using this method:

```
public String getRemoteUser()
```

This method returns the name of the user making the request. This information can be used for authorization purposes, as well as to give information about each client, which can be used over time to save an individual's preferences. It returns null if the client has not been authenticated.

For the servlet to find out what the client has requested, it accesses request parameters associated with the request. Every request a servlet receives can have any number of request parameters. These parameters are always name-value pairs. The servlet gets its request parameters as a part of the query string for GET requests, or as encoded data for POST requests. However, they retrieve their parameters in the same way, by using these methods from `ServletRequest`:

```
public String getParameter(String name)
public String[] getParameterValues(String name)
```

In addition, a servlet can access parameter names using the following method of `ServletRequest`, which returns all the parameter names as an enumeration of String objects:

```
public Enumeration getParameterNames
```

An HTTP request includes extra file information to indicate a file on the server. Servlets do not need the support, but a CGI program cannot interact with its server and does not receive the path parameter. To overcome this deficiency, there is a need to support extra path information, which is encoded in the URL. A servlet can access this information and can translate it into the real path using these methods `ServletRequest`:

```
public String getPathlnfo()
public String getPathTranslated()
```

The path information is only useful for a servlet if it knows the actual location of the file. For example the path /index.html isn't very useful until we translate it to the actual file system location the file in the path information, for example:

```
/Repository/WebApplication/Tracking/index.html
```

The servlet can use several methods to find out exactly what file or servlet the client requested. We can use the getRequestURL() method of `HttpUtils`:

```
public static StringBuffer getRequestURL(HttpServletRequest
request)
```

And the getRequestURI() and getServletPath() methods of `HttpServletRequest`:

```
public String getRequestURI()
public String getServletPath()
```

A servlet usually needs only the request URI (which can be thought of as a URL, without scheme, host, port, and query string for a normal servlet). For servlets in a chain, the request URI is always that of the first servlet in the chain. The getServletPath() method is used when it is enough to know the servlet name under which it was invoked.

The details about the manner in which the request was made can also be made available by using the Scheme() and getProtocol() methods of ServletRequest:

```
public String getScheme()
public String getProtocol()
```

With the help of the getScheme() method the servlet can find out whether the request was made over a secure connection (as indicated by the scheme "https") or was an insecure request (as indicated by the scheme "http"). The other schemes may include "ftp", "jdbc", or "rmi". The servlet uses getProtocol() to get the protocol and version number. The protocol and version number are separated by a slash, for example "HTTP/l.0" or "HTTP/1.1".

The HTTP headers also provide additional information about the request. The server deals directly with most of the details of the headers. For example, when a server receives a request for a restricted page, it checks that the request includes an appropriate Authorization header. However, servlets may want to read a few headers on occasion. A header value can be retrieved as a String, a long, or an int using these methods of HttpServletRequest:

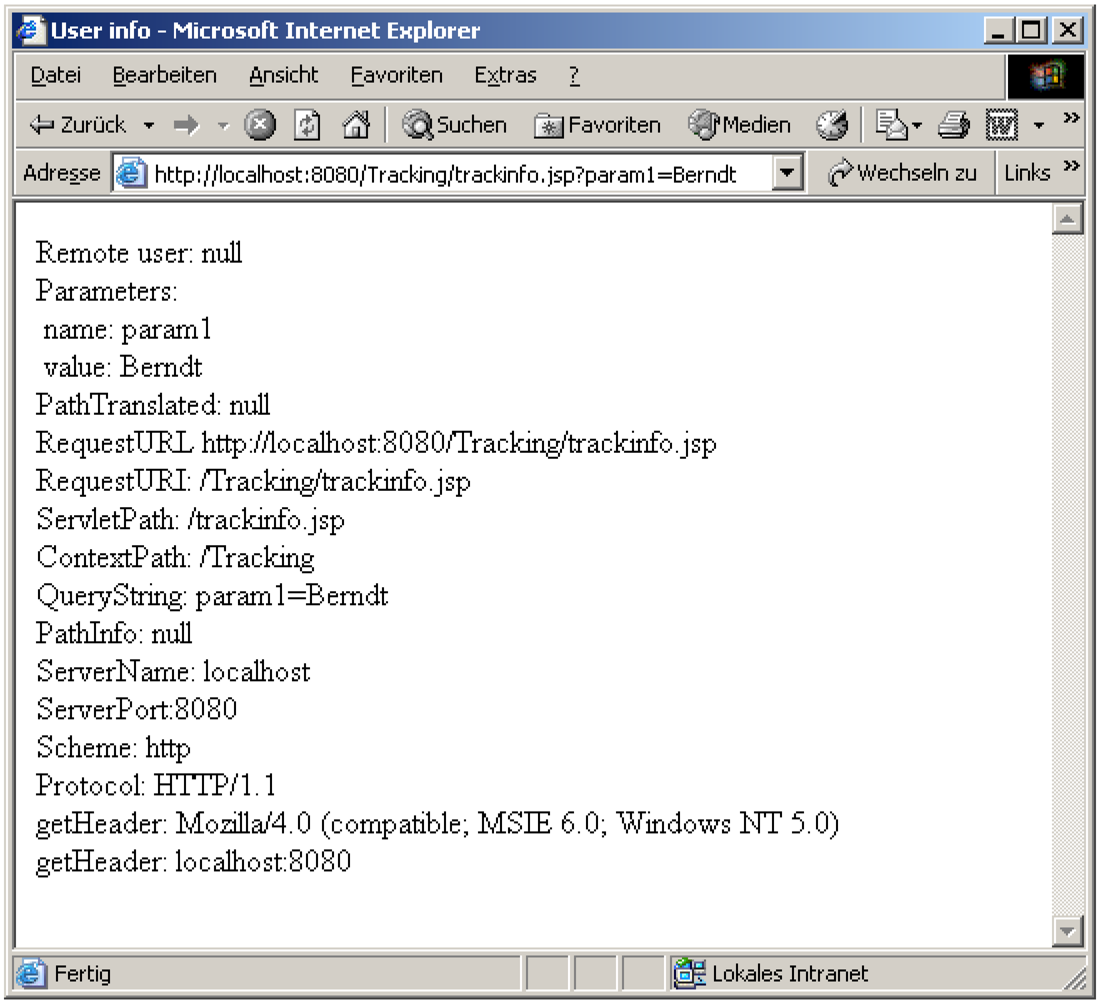
```
public String getHeader(String name)
public long getDateHeader(String name)
public int getIntHeader(String name)
```

Some of the headers that a servlet may want to read are:

- Accept: Specifies the media (MIME) type the client prefers to accept A servlet can use this header to determine the type of content to return. For example, Accept: text/*

- User-Agent: Gives information about the browsers name and version that can be used to customize its response
- Authorization: This includes the username and password (the client's authorization to access the resource), which can be used for custom authorization

I implemented these calls in a JSP and if we call it we get something like the following response:


User info - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://localhost:8080/Tracking/trackinfo.jsp?param1=Berndt
Wechseln zu Links
Fertig
Lokales Intranet

```
Remote user: null
Parameters:
 name: param1
 value: Berndt
PathTranslated: null
RequestURL http://localhost:8080/Tracking/trackinfo.jsp
RequestURI: /Tracking/trackinfo.jsp
ServletPath: /trackinfo.jsp
ContextPath: /Tracking
QueryString: param1=Berndt
PathInfo: null
ServerName: localhost
ServerPort:8080
Scheme: http
Protocol: HTTP/1.1
getHeader: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
getHeader: localhost:8080
```

Figure 1 – User Information Read From a Servlet

Visitor Tracking Using Clickstream

One of the components of the open source project for providing enterprise class J2EE applications and components, OpenSymphony, is Clickstream. It is a small utility to show the users on your site and provide details on where they have been.

Download the beta component at http://www.opensymphony.com/clickstream/

Clickstream allows you to track the 'traffic paths' across your site. It starts tracking when a users' session begins and tracks information with each click the user makes. It is designed to log the complete click stream to a file or PrintStream when the users' session ends. It also tries to discover whether the user is a bot, and filters accordingly - currently over 250 bots are detected.

(A bot is a piece of software used to dig through data. They are commonly employed by search engines to compile their databases of sites on the Web.)

Clickstream itself is still quite basic as the code is very simple and contains:

- A filter to track requests - ClickstreamFilter
- A class to check for bots - BotChecker
- A class to listen for the Servlet 2.3 events - ClickstreamLogger
- A bean to store details of the click stream – Clickstream

Quick Introduction to Servlet Filters

As Clickstream relies on servlet filter technology we should take a quick look at them. Servlet filters are a new addition to the Java™ Servlet 2.3 specification. Filters are Java classes that can intercept requests from a client before they access a resource; manipulate requests from clients; intercept responses from resources before they are sent back to the client; and manipulate responses before they are sent to the client.
Filters have a wide array of uses; the Servlet 2.3 specification suggests the following uses:

- authentication filters
- logging and auditing filters
- image conversion filters
- data compression filters
- encryption filters
- tokenizing filters
- filters that trigger resource access events
- XSL/T filters that transform XML content
- MIME-type chain filters

A filter is simply a Java class that implements the `javax.servlet.Filter` interface. The `javax.servlet.Filter` interface defines three methods:

- `public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`
- `public FilterConfig getFilterConfig()`
- `public void setFilterConfig(FilterConfig filterConfig)`

It is the containers responsibility to create a `javax.servlet.FilterConfig` object and pass it to the filter during initialization. The `javax.servlet.FilterConfig` object can be used to

- retrieve the filter name (as defined in the deployment descriptor),
- retrieve the initial parameters (as defined in the deployment descriptor), and
- get a reference to the ServletContext object the user is calling from

The `setFilterConfig()` method can be used to capture the object into an attribute of the filter. The `doFilter()` method is where the filter's work is done, in which you can parse the user's request; log statistics to a file; manipulate the response back to the client; and so on.

The filters in a web application are defined in the deployment descriptor of the web application, the `web.xml` file. Filters are defined and then mapped to a URL or Servlet, in much the same was as Servlet is defined and then mapped to a URL pattern. We will see this when we install Clickstream.

For more information on servlet filters, please read Sun's "The Essentials of Filters" at: http://java.sun.com/products/servlet/Filters.html.

The article is also available in PDF format http://java.sun.com/products/servlet/Filters.pdf .

Clickstream Installation

The installation is straightforward:

1. Create a new Web Application using Jaguar Manager and give it the name: Tracking
2. Create a WEB-INF in your %Jaguar%/Repository/WebApplication/Tracking directory
3. Copy the ClickStream classes into the \WEB-INF\classes directory of your web application:

```
\WEB-INF\classes
   Clickstream.class
   ClickstreamFilter.class
   ClickstreamLogger.class
   BotChecker.class
```

4. We also need to place two JSP files in the application directory, clickstreams.jsp and viewstream.jsp.
5. Now, we have to configure a filter, a filter mapping and listener for our WebApplication:

```
<filter>
<filter-name>ClickstreamFilter</filter-name>
<filter-class>ClickstreamFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>Clickstream Filter</filter-name>
<url-pattern>/*.html</url-pattern>
</filter-mapping>
<listener>
<listener-class>ClickstreamLogger</listener-class>
</listener>
```

This currently tracks any click on HTML files. If you want to track other clicks (for example JSP, TXT or PDF files), just add more filter mappings.

6. Finally, add to the WebApplication two simple html files (call them `test.html` and `test1.html`), which we will call and test that ClickStream tracks them.

ClickStream and EAServer 4.1

For ClickStream to work on EAServer 4.1, we need to make some modifications to the downloaded files. Essentially, we need to make the software compatible with the Servlet 2.3 specification.

We need to update the ClickstreamFilter class to the latest API by renaming the `setFilterConfig()` method to `init()` and adding an empty `destroy()` method:

```
public void init(FilterConfig filterConfig) {
this.filterConfig = filterConfig;
}
public void destroy() {}
```

As it is always a good practice to have classes in a package we put our Clickstream classes in a package of their own, and then we import this package into each JSP page. Add the following line to each Clickstream class:

```
package Clickstream;
```

Then put the source files into a Clickstream directory, before recompiling them. Then add the following line to both JSP pages:

```
    page import="Clickstream.*"
```

We will ignore `web.xml` and configure ClickStream using Jaguar Manager.

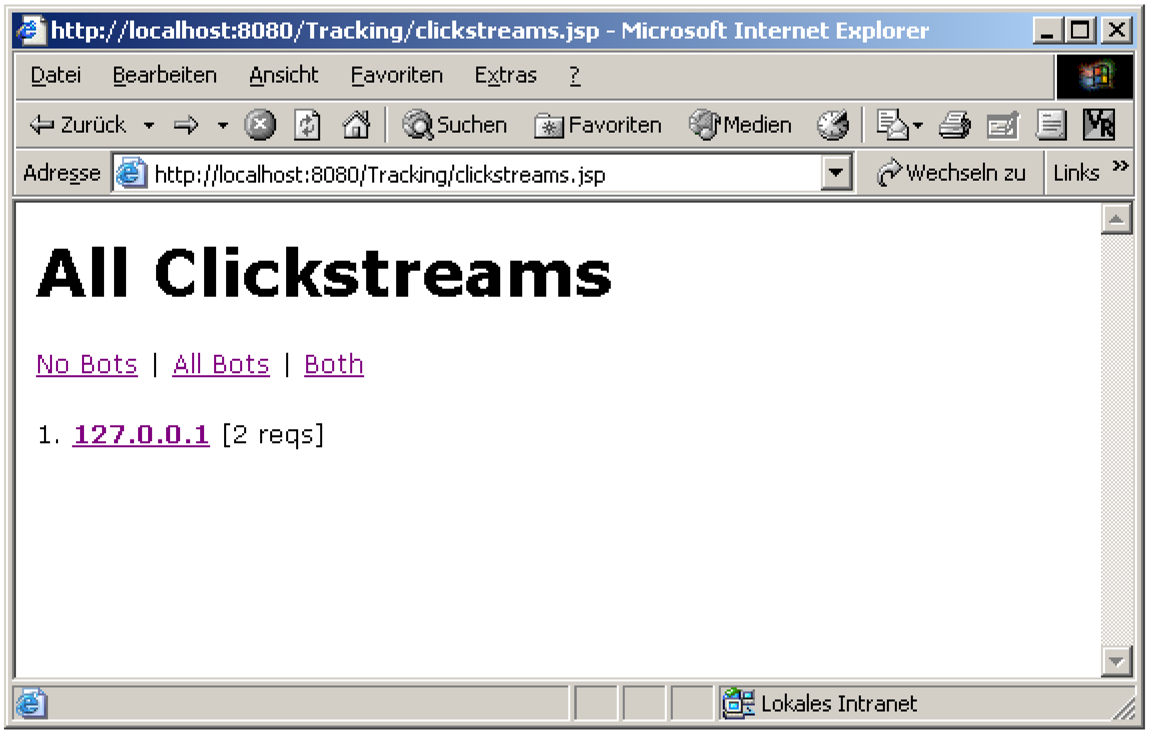
If you don't want to make these changes, you can find the updated files and WebApplication on the [Sybase Developer Network (SDN)] *page you downloaded this white paper from.*

Now test the shopping cart web application. Use the URLs

```
    localhost:8080/Tracking/test.html
    localhost:8080/Tracking/test1.html
```

Then to see your requests, simply navigate to `clickstreams.jsp`

You can see who is currently on this site here, and even view details about a particular stream:

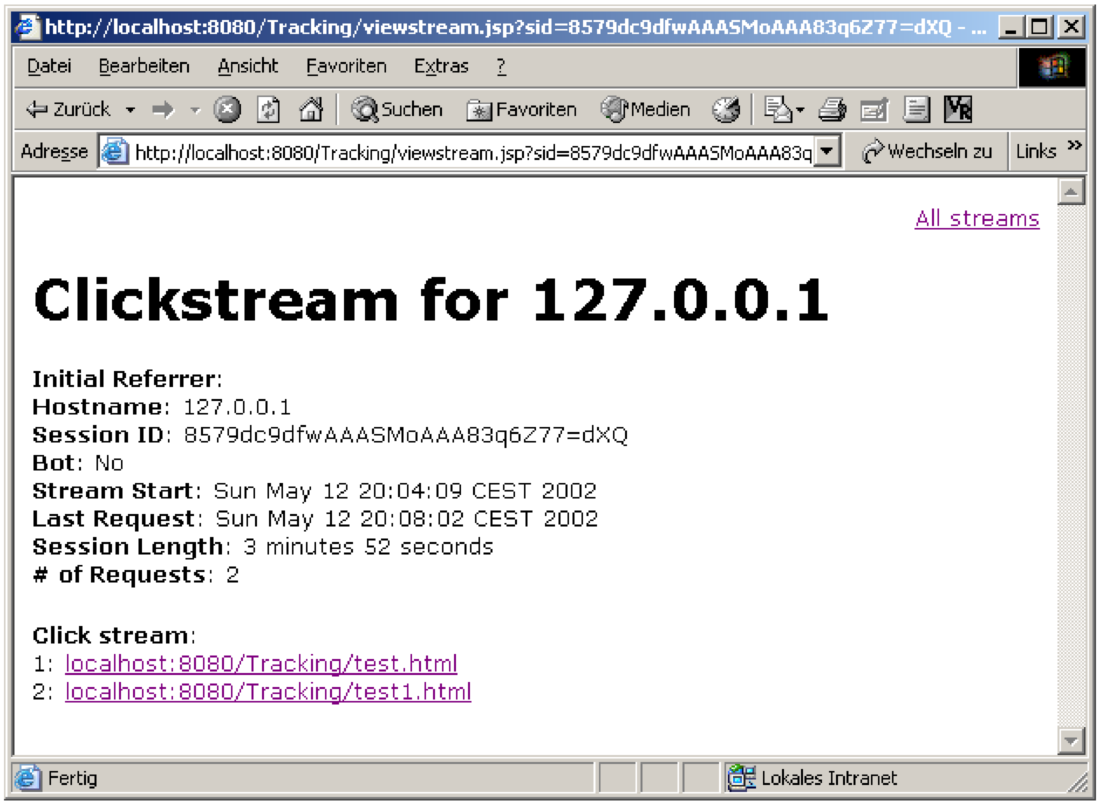

http://localhost:8080/Tracking/clickstreams.jsp - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://localhost:8080/Tracking/clickstreams.jsp
Wechseln zu Links
Lokales Intranet

All Clickstreams

No Bots | All Bots | Both

1. 127.0.0.1 [2 reqs]

Figure 2 – A Request From Address 127.0.0.1


http://localhost:8080/Tracking/viewstream.jsp?sid=8579dc9dfwAAASMoAAA83q6Z77=dXQ - ...
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://localhost:8080/Tracking/viewstream.jsp?sid=8579dc9dfwAAASMoAAA83q
Wechseln zu Links
All streams
Fertig
Lokales Intranet

Clickstream for 127.0.0.1

Initial Referrer:
Hostname: 127.0.0.1
Session ID: 8579dc9dfwAAASMoAAA83q6Z77=dXQ
Bot: No
Stream Start: Sun May 12 20:04:09 CEST 2002
Last Request: Sun May 12 20:08:02 CEST 2002
Session Length: 3 minutes 52 seconds
# of Requests: 2

Click stream:
1: localhost:8080/Tracking/test.html
2: localhost:8080/Tracking/test1.html

Figure 3 – Details On User

As you can see, Clickstream can provide you with potentially very useful information about your site that would otherwise be tedious to collect.

Sadly, Clickstream is not a work-in-progress anymore, so you will have to enhance the software on your own.

Summary

In this article, I showed you how you could start to keep track of users. We began by looking at the type of information that a client makes available during a request, and how we can access this information. Then, we looked at how we can use software such as Clickstream to keep track of users as they access our site. This information, if properly analyzed, can help us to better design our site to fulfill the needs of our visitors. Now go and build upon this simple example and keep having fun with Sybase EAServer.

About the Author

Berndt Hamboeck is a senior consultant for Sybase Austria. He is an EASAC, SCAPC8 and CSI and SCJP2.

Sybase Incorporated
Worldwide Headquarters
5000 Hacienda Drive
Dublin, CA 94568-7902, USA
Tel: 1-800-8-Sybase, Inc.
Fax: 1-510-922-3210
www.sybase.com