FreeFrameWork White Paper – Session Management Methodology

Draft – Not reviewed by the FreeFrameWork Board of Directors.
Steve Southwell, 8/2/2000

What is Session management?

As it applies to applications for the web, Session management is the way in which a WebSpeed agent, which must serve many clients from one moment to the next, quickly determines who is making a request and retrieves a record of their session. This session record, which may contain information that helps the program customize content or authenticate users and their permissions, is then used by the program as a key to other details.

How is Session Management achieved?

We keep track of the user's session by tagging them with a sessionID upon session startup. The program detects session startup when it receives any request from a user that does not carry with it a "token" containing a currently valid sessionID.

Upon session startup, these things happen:

- A unique and random sessionID is generated.
- A WebSession record is created in the database.
- Two WebHit records are created: One labeled "0" (zero) indicates the start of the session, and serves as a "dummy" record so that a previous webHit buffer is always available on every hit, including the first one. Another WebHit record is created to store the context of the current hit.
- A flag is set to notate that his is a brand-new session.
- When the HTTP header is output, a cookie is sent to the user if the brand-new-session flag is set. In this way, the session cookie is only set once, and will not annoy users that choose to either turn them off or be warned about them. (Note that a cookie is not enough to maintain state, other steps are mentioned below.)

On each and every hit, these things happen:

- Cleanup of old WebSession and WebHit records for up to 1 second. (Usually way less than that)
- The program checks for the contents of the "stateinfo" field, which may come in as part of the query_string variable on a GET, or in a hidden field on a POST.
- If no "stateinfo" field is detected, it searches for an incoming cookie called "SafeSessionID", which is the cookie set upon initial session startup.
- Using either the cookie or the stateinfo field, whichever one it found first, it will attempt to locate a WebSession record.
- If the WebSession is expired, or if no cookie or stateinfo field was found, then a new session is started.
- Once a WebSession is found or started, the program loads two WebHit buffers: PrevWebHit, and ThisWebHit.

How is Session Management Implemented?

In a nutshell, contest management requires us to retrieve a user's session data based on a SessionID token that gets sent to the server on each hit. Although the cookie method mentioned above is a simple and widely used method for doing this, it is unreliable due to the fact that a consistent 6% of internet users turn them off or refuse them.

In order to get around this limitation, we must make sure that every time the user posts a form, or clicks a link within the dynamic portion of our website, that they send us the token. To do this, we simply insert the token into every link, and every form. In order to make this easier, two special preprocessors have been

defined that can be easily inserted into your code: urlStateInfo and hiddenStateInfo. Each webobject that uses FFW's session management scheme must include urlStateInfo in each internal link, and hiddenStateInfo in each form. For example:

```
<a href="something.w?foo=bar&`{&urlStateInfo}`">Your link</a>

or

<script language="javascript">
. . .
window.location.href = 'something.w?foo=bar&`{&urlStateInfo}`';
. . .
</script>
```

Notice in the example above, that you must use a query_string delimiter (either ? or & as the case may be) prior to inserting the {&urlStateInfo} (which must be enclosed in backticks, if used in ESS)

```
<form action="whatever.w">
. . .
`{&hiddenStateInfo}`
. . .
</form>
```

Notice above how hiddenStateInfo is enclosed in backticks and placed anywhere in the form. I prefer to place it at the top of the form, so that I can easily find it and make sure that I put it in. However, if you are using any javascript validation that looks at the form elements in terms of element number rather than element name, then putting it at the end is a better idea.

Include files

All of the statekeeping logic mentioned above, as well as the preprocessor definitions for hiddenStateInfo and urlStateInfo are included in the set of method libraries distributed with FFW's session management system.

In ESS (Embedded Speedscript) files, simply insert the following include file reference at the top of your .html program:

```
<script language="speedscript">
. . .
&global-define web-file yourfilename
{ lib/esslib.i }
. . .
</script>
```

Note that this must be included in your code above almost everything else.

In CGI wrapper programs and Mapped HTML programs you must include lib/ffwebstate.i right after the src/web/method/wrap-cgi.i or src/web/method/html-map.i
inclusion. For example:

```
/* Included Libraries --- */
{src/web/method/wrap-cgi.i}
{lib/ffwebstate.i}
```

In both CGI wrapper and HTML-Mapped programs, you must modify your process-web-request procedure so that it calls Get-Last-State prior to anything else. For example:

```
PROCEDURE process-web-request :
/*------------------------------------------------------------
  Purpose:     Process the web request.
  Parameters:  <none>
  Notes:
------------------------------------------------------------*/
   RUN Get-Last-State.
```

Glossary

LastWebHit – Buffer for the WebHit table which contains the WebHit record of the hit which either came immediately prior to the current hit, or the hit which created the page from which the user either submitted a form or clicked a link to come to this hit. If you need to retrieve some sort of previously calculated value, you could obtain it from this buffer.

Session - (see also WebSession) a given set of interrelated interactions between a user and a web application in a given timeframe. Sessions remain active as long as the user continues to interact with the system. If the user fails to interact with the system for a given period of time, which can be configured, then the session will "time-out" and will no longer be usable, although it will remain in the database for logging purposes.

ThisWebHit – Buffer for the WebHit table which contains the WebHit record for the current hit. If you need to save information about the user's current hit, just assign it to this buffer.

WebHit – Table for storing information on each hit of a session. For instance, if a user came to your site, followed a link, and then followed another link off of that page, there would be one WebSession record, and 3 hits + 1 pseudo hit = 4 WebHit Records.

WebSession – Table for storing general information about a web user's session with your application. WebSession records persist for generally about a month (this is configurable) before they are erased from the database to make more room. Although the records stay in the database, they are generally only active for the period of time in which the user is actively interacting with the system.

WebSurfer – One individual who has had at least one, but possibly many sessions with your web application. This table is a master record for information that needs to survive from one session to the next. As such, a webSurfer must be verified as being the same person before their personal data can be reliably used. Depending on your setup, you can choose to either actively or passively profile websurfers, storing free-form data about them for later use in interface customization or for marketing purposes.