

## Maintaining State Server-side solution

Gordon College  
CS374  
Internet Programming

## State Comparisons

	Client-Side Methods			Server-Side Methods		
	Hidden Field	URL Parameter	Cookie	Server Memory	File	Database
Suitable for Long-Term Storage			x		x	x
Survives Server Restarts	x	x	x	Perhaps	x	x
Survives Host Restarts	x	x	x		x	x
Automatic Expiration	x	x	x			
Hackable by Client	x	x	x			
Potential for Exposure to Other Clients	High	High	High	Low	Low	Low

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

State ID given to client – either in hidden field on form or as link.

Format data in state file for easy access.

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state  
State ID given to client – either in hidden field on form or as link.

Format data in state file for easy access.

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state  
State ID given to client – either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state  
State ID given to client – either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

hash-in-a-file – use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Must eliminate obsolete state files.

### Using a server-side file solution:

```

S Hash-in-a-file???
H
Dl
St
St
<a
Fc
ha
St
M
}
return %hash;
### writing
open(OUTFILE, ">$dir$filename") or &errorPage("Error writing to state file.");
foreach $key (keys %states) {
    $value = &URLEncode($states{$key});
    $name = &URLEncode($key);
    print OUTFILE "$name=$value\n";
}
### reading
my %hash = ();
foreach $line (@array) {
    chomp $line;
    my ($key, $value) = split(/=/, $line, 2);
    $key = &URLdecode($key);
    $value = &URLdecode($value);
    $hash{$key} = $value;
}

```

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state

State ID given to client - either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

hash-in-a-file - use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state

State ID given to client - either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

hash-in-a-file - use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Randomly generate name of file - 62 characters per char with 32 characters

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state

State ID given to client - either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

hash-in-a-file - use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Randomly generate name of file - 62 characters per char with 32 characters

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state

State ID given to client - either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

hash-in-a-file - use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Randomly generate name of file - 62 characters per char with 32 characters

Must eliminate obsolete state files.

### Using a server-side file solution:

**state file** - each session has a corresponding state file to store state data during the session.

#### Hurdles:

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf98GV76Y8H8kjoI.state

State ID given to client - either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz...">...</a>

Format data in state file for easy access.

hash-in-a-file - use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Randomly generate name of file - 62 characters per char with 32 characters

Must eliminate obsolete state files.

**Using a server-side file solution:**

**state file** - each session has a corresponding state file to store state data during the session.

**Hurdles:**

Different state file for each session.

Big random number for ID - C9JzoLZh998LKJtyf198GV76Y8H8kjoI.state

State ID given to client - either in hidden field on form or as link.

<input type="hidden" name="id" value="C9Jz..." /> or

<a href="myprogram.cgi?id=C9Jz..."></a>

Format data in state file for easy access.

hash-in-a-file - use name/value pairs like what is used in a hash

State file names should be unique so that a new session does not overwrite the state file for a session in progress.

Randomly generate name of file - 62 characters per char with 32 characters

Must eliminate obsolete state files.

Manually, with a script, or allow the Perl program to handle it based on

constraints:

\$cache\_limit = 300; # max number of files in cache

\$file\_life\_span = 1; # kill files older than one day

**Using a server-side file solution:**

```
my ($dir, $cache_limit, $file_life_span) = @_;

S opendir(DIR, $dir) or &errorPage("Error Logging On.");
my @files = readdir(DIR);
closedir(DIR);

if($#files >= $cache_limit {
  foreach $file (@files) {
    if((-f "$dir$file") && ((-M "$dir$file") > $file_life_span)) {
      unlink "$dir$file"; #delete the file
    }
  }
}

opendir(DIR, $dir) or &errorPage("Error Logging On.");
@files = readdir(DIR);
closedir(DIR);

if($#files >= $cache_limit {
  # should generate e-mail message to warn administrator (see S
  &errorPage("Site busy. Please try again later.");
}

$cache_limit = 300; # max number of files in cache
$file_life_span = 1; # kill files older than one day
```

**&generate\_random\_string**

-- returns a session ID (of specified length)

Example use:

```
$sessionId = &generate_random_string(32);
```

It's then easy to build the full name of a state file:

```
$filename = "$sessionId.state";
```

**&write\_state**

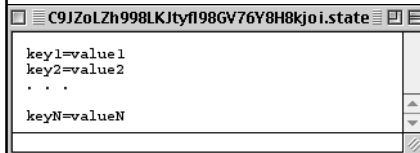
-- writes a hash to a state file

-- creates a new state file or overwrites an existing one

Example use:

```
&write_state($stateDir, $sessionId, %stateHash);
```

The hash is written to the file in the following format:



**&read\_state**

-- returns a hash containing the data found in a state file

Example use:

```
%stateHash = &read_state($stateDir,$sessionId);
```

URL-encoded in the state file.

- keep unwanted characters (like =) in the data from interfering with the structure (delimiting characters) of the state file.

- potential security risk which can arise from a cleverly placed \n character in the state data.

- Perl's directory functions are used to read in the names (as strings) of all the state files found in the cache.

Example:

```
opendir(D, "path/to/cache/directory/");
@allfiles = readdir(D);
closedir(D);
```

- We can then easily determine the size of the cache  `$#allfiles`.
- Moreover, the files in the cache can be individually examined by looping over the `@allfiles` array.

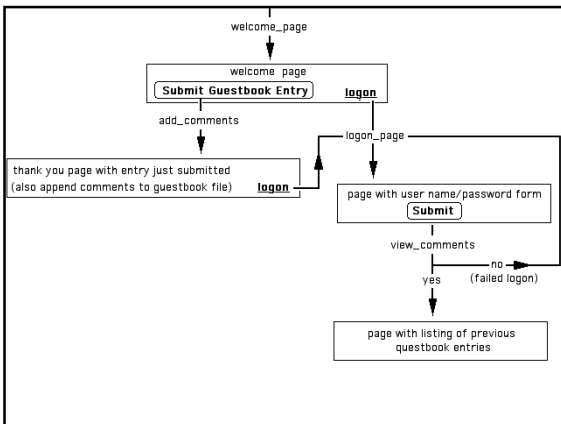
### Perl's file test operators.

To test the "age" of state files:

**-M filename** returns days as a real number since the file was last modified

To make sure that what we have is actually a state file:

**-f filename** returns false (0) if it is not a file true (1) if it is



**Logged on state** -- the state of prolonged access to private pages (functions) in the application.

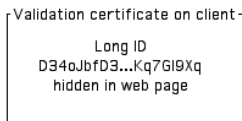
- The main idea is that after one successful logon, the entire private portion of the application is available.
- That might mean one logon allows you to call any of several private app logic functions.
- Or one successful logon allows to call one private app logic function over and over again.

### Here's how to implement logged on state:

1. Upon successful logon, create a session ID and state file.



2. Hide the session ID in every page subsequently returned to the user.



### The result:

- The session ID serves as a **validation certificate**.
- State file contains information relevant to status of the logged on user.
- Successful login is the only way to obtain a session ID pointing to a state file

- Then, each private function in the app logic needs to reject a user when called with out a validation certificate for logged on state.

```

sub some_private_function {
  %stateHash = &read_state($stateDir,$sessionID);
  if ($stateHash{"access"} ne "admin") {
    &some_public_function("Sorry buddy!");
    exit;
  }

  # else, the user has logged on as admin at some
  # point so print the private page
}

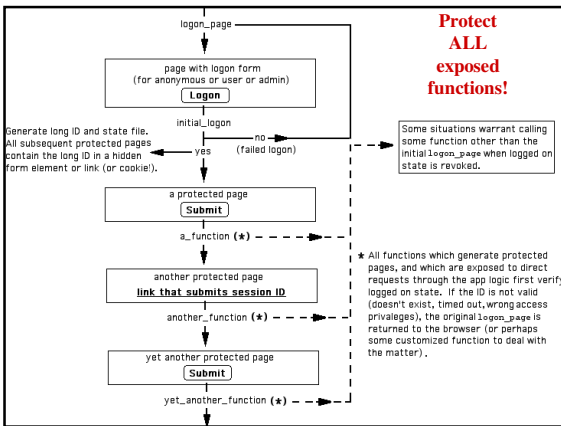
```

```

## app logic for admin.cgi

if($formHash{"request"} eq "initial_logon") {
  &initial_logon; ## private
                    ## initiate logged on state
}
elseif($formHash{"request"} eq "add_user") {
  &add_user;      ## private
                    ## verify logged on state
}
else {
  &logon_page;
}

```



**Note on validation certificates:** The mere existence of a state file can be used to verify logged on state.

```

if (!( -e "path/to/sessionID.state" ) ) {
  &logon_page; # revoke session
  exit;
}

```

- Here, `-e` tests the existence of the file, returning true or false.