

Session Variables in CMRL

CMRL Version 1.0



Contents

1	Introduction	3
2	The Problem	3
3	The Solution	3
4	The <code><get></code> Tag	5
5	The <code><input></code> Tag	6
6	Summary	7

1 Introduction

Session variables in CMRL are used to store information on a per-user basis (e.g. user preferences, usage history, account identification, etc.). Session variables are similar to cookies on the web: they make it easy to customize services for individual users.

This document describes session variables in CMRL. The document assumes that you have some familiarity with DOTGO and CMRL (obtained, e.g., by reading [A Brief Introduction to DOTGO](#)).

2 The Problem

Suppose you were to implement an engine under the internet domain name “example” and the match “weather” that returns a weather forecast based on zip code. The engine might be referenced by the following CMRL fragment:

```
<match pattern="weather">
  <engine href="http://example.com/cgi-bin/weather.cgi"/>
</match>
```

where “weather.cgi” is a web-based CGI program that returns a weather forecast given a zip code. The first time a particular user accesses the service, the user must, of course, provide a zip code. But what about the next and subsequent times the same user accesses the service? Wouldn’t it be nice if the zip code could default to the previous value if it is not supplied?

3 The Solution

There is a way to implement this: session variables. Session variables are used to store information—like zip codes—on a per-user basis. Suppose the web-based CGI program “weather.cgi” consists of the the following Perl program (together with some unspecified function “forecast” that returns a weather forecast given a zip code):

```
#!/usr/bin/perl

use CGI qw(:standard);
use strict;

my $argument = param("sys_argument");
my $last_zip_code = param("last_zip_code");

$argument =~ m/(\d\d\d\d\d)/;
my $current_zip_code = $1;

my $content;
if ($current_zip_code) {
  $content = forecast($current_zip_code);
  $last_zip_code = $current_zip_code;
}
```

```

} elsif ($last_zip_code) {
    $content = forecast($last_zip_code);
} else {
    $content = "Please supply a five-digit zip code--try again";
}

print header;
print <<EOF
<block>
    <set name="last_zip_code">$last_zip_code</set>
    <message>
        <content>$content</content>
    </message>
</block>
EOF
;

```

With this construction, the previous value of the zip code—if there is a previous value of the zip code—is stored in the session variable “last_zip_code.” The first time a particular user accesses the service, the query “example weather” produces the response “Please supply a five-digit zip code—try again,” whereas the query “example weather 90210” produces a weather forecast for zip code 90210 as a response. Once a zip code is specified, the next and subsequent times the same user accesses the service, the query “example weather” produces a weather forecast for zip code 90210 as a response.

So what’s going on? The session variable “last_zip_code”—if it exists—is posted to the engine every time the engine is called. The engine assigns the value of the session variable “last_zip_code” to the Perl variable “\$last_zip_code” in the line

```
my $last_zip_code = param("last_zip_code");
```

of the Perl program. If no zip code is supplied, and if there is a previous value of the zip code, then the engine uses the previous value of the zip code to produce the weather forecast. The engine sets the value of the session variable “last_zip_code” every time it is called using the <set> tag within the <block> tag. The <block> tag is a CMRL terminating node that allows various “helper” tags to be associated with another terminating node. The <block> tag must contain exactly one terminating node (which in this case is the <message> tag) and may contain various other tags, including zero or more <set> tags (and zero or one <keywords> tags, which are beyond the scope of this document). The <set> tag must contain the attribute name, which is the name of the session variable to be set (in this case “last_zip_code”), and must contain a text string, which is the value to be assigned to the session variable (in this case the value of the Perl variable “\$last_zip_code”).

Session variables are stored *per user and per CMRL document*. When a particular user accesses a particular CMRL document, all session variables defined for that user and that CMRL document are available to the CMRL document (and are posted to engines referenced by the CMRL document).

4 The <get> Tag

The <get> tag is used to get the value of a session variable from within CMRL. Consider the following CMRL fragment under the scenario described in sections 2 and 3:

```
<match pattern="show default">
  <message>
    <content>Default zip code: <get name="last_zip_code"/></content>
  </message>
</match>
```

Under this construction, if the session variable “last_zip_code” does not exist, then the the query “example show default” produces the response “Default zip code: ,” whereas if the session variable “last_zip_code” exists, then the query “example default” produces the response “Default zip code: 90210.”

5 The <input> Tag

The <input> tag can be used to set the value of a session variable. Consider the following CMRL fragment under the internet domain name “example”:

```
<match pattern="name">
  <message>
    <content>Reply with your first name</content>
    <input name="first_name">
      <message>
        <content>Thanks <get name="first_name"/></content>
      </message>
    </input>
  </message>
</match>
```

Under this construction, the query “example name” produces the response “Reply with your first name.” The reply “Sally” then produces the response “Thanks Sally.”

The <input> tag is used within a <message> tag. The <input> tag must contain exactly one terminating node (which in this case is another <message> tag) and may contain the attribute name, which is the name of the session variable to be set. If a <message> tag contains an <input> tag, then the reply sent by the user is treated differently than normal; in particular, it is passed directly to the terminating node of the <input> tag as an argument. If the terminating node of the <input> tag is a <message> tag, then the reply has no effect (since a <message> tag cannot interpret an argument). But if the terminating node of the <input> tag is an <engine> tag, then the reply is passed directly to the engine as an argument, with no other processing by the system. In this way, the <input> tag can be used to direct arbitrary user input to an engine. If the name attribute of the <input> tag is specified, then the value of the session variable specified by the name attribute is set to the reply. In this way, the <input> tag can be used to store arbitrary user input in a session variable.

Two or more <input> tags can be nested, allowing multiple session variables to be set sequentially. For example, consider the following CMRL fragment:

```
<match pattern="name">
  <message>
    <content>Reply with your first name</content>
    <input name="first_name">
      <message>
        <content>Now reply with your last name</content>
        <input name="last_name">
          <message>
            <content>Thanks <get name="first_name"/>
              <get name="last_name"/></content>
          </message>
        </input>
      </message>
    </input>
  </message>
</match>
```

Under this construction, the query “example name” produces the response “Reply with your first name.” The reply “Sally” then produces the response “Now reply with your last name.” The reply “Jones” then produces the response “Thanks Sally Jones.”

6 Summary

By storing information on a per-user basis, session variables make it easy to customize services for individual users. The next step is to try it for yourself.