

## Dynamic HTTP Redirection (HTTP 302)

The following example script performs HTTP Redirection following an HTTP 302 error.

Use the following link to download and import the XML for the entire HTTP Redirection Script or copy and paste only the script code.

Note: XML can be imported using the SOASTA CloudTest > Repository toolbar > Import icon. Refer to Script to learn how to add a script to a test clip and test composition.

When recording an HTTP scenario, some HTTP 302 responses may be recorded in the traffic. If this recording is converted to a clip and played back unaltered, the requests used will faithfully use the recorded content. Load tests will typically want requests following the 302 response to redirect subsequent requests to the site specified by the Location header in the response. Frequently this kind of redirection is done in order to spread load out amongst a set of servers.

The following script example is meant to demonstrate how to recognize redirection automatically. A script like this can be placed immediately after any message in a clip that would normally generate a 302 response.

```
var pos = origText.indexOf("HTTP/1.1 302 Found");
if (pos == -1)
{
    $context.result.postMessage($context.result.LEVEL_INFO, "did not find 302");
}
else
{
    //now get the value of the location header
    // and then the protocol following
    var firstPos = origText.indexOf("Location:");
    firstPos = origText.indexOf("http", firstPos + 1);

    var lastPos = origText.indexOf("\n", firstPos + 1);

    var location = origText.substring(firstPos, lastPos);

    //now we need to separate this into host name and service path
    var doubleSlashPos = location.indexOf("//");
    firstSlashPos = location.indexOf("/", doubleSlashPos + 2);

    //extract out the host name and the service path from the location
    var hostName = location.substring(doubleSlashPos + 2, firstSlashPos);
    var servicePath = location.substring(firstSlashPos);

    //set the host name and service path for the target of the next message

    $context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("HostName", hostName);

    $context.currentItem.nextItem.target.systemPropertyList.setPropertyValue("ServicePath", servicePath);
}
```

The script first gets the message that immediately precedes the script. This is the message with the 302 response (line 6). The following line gets the content of the response (line 7).

```
var msg = $context.currentItem.previousItem;
var origText =
msg.getResponse(msg.RESPONSE_TEXT);
```

Next, the script checks to make sure there really was a 302 response and reports out if there was not. Line 10 uses the JavaScript "indexOf" method to determine if this was a 302 response.

```
var pos = origText.indexOf("HTTP/1.1 302
Found");
```

Assuming there was a 302 response, lines 18 through 35 process the response text. Line 18 finds the position of the location header, then uses that position to find the beginning of the URL to which we are being redirected.

```
var firstPos = origText.indexOf("Location: ");
firstPos = origText.indexOf("http", firstPos
+ 1);
```

Once we know where the beginning of the URL is, we need to find the end of the location header content. This is accomplished in line 21, which finds the end of the line following the URL start. Line 23 uses these positions to extract out the location URL from the message response content.

```
var lastPos = origText.indexOf("\n", firstPos
+ 1);
//
var location = origText.substring(firstPos,
lastPos);
```

Because we will eventually need the host name and the service path as two separate strings, we now need to parse the URL. To get the host name, we need the portion of the URL between the "http://" (or "https://") and the first single slash. Line 26 finds the "://" and then uses that position to get the first single slash in line 27.

```
var doubleSlashPos = location.indexOf("://");
firstSlashPos = location.indexOf("/",
doubleSlashPos + 2);
```

We now have enough string position information to extract out the host name (line 30) and the service path (line 31).

```
var hostName =
location.substring(doubleSlashPos + 2,
firstSlashPos);
var servicePath =
location.substring(firstSlashPos);
```

Finally, we use the SOASTA-provided methods to set the system properties "HostName" and "ServicePath" for the target of the next message. This is the message that needs to be redirected. Because all messages with the same target share one target "object", setting the system properties for the target will redirect all subsequent messages with that same target.

```
$context.currentItem.nextItem.target.systemPro
pertyList.setPropertyValue
("HostName", hostName);
$context.currentItem.nextItem.target.systemPro
pertyList.setPropertyValue
("ServicePath", servicePath);
}
```

Line 34 gets the current item (the script) and then the next item (the message following the script) and then gets the target for that message. Each target has a system property list ("systemPropertyList") object and the system property list permits the setting of system properties. Here we set the target system property "HostName". Line 35 does the same for "ServicePath".

Other system properties are:

<b>System Property</b>	<b>Value</b>
"URL"	target URL
"HostName"	host name
"ServicePath"	service path (everything after the host name)
"Port" target	Port
"UseSSL"	"true" or "false" to control use of secure socket layer
"UserName"	secure socket Layer user name
"Password"	secure socket Layer password
"MaximumConnectionsPerHost"	maximum number of http connections (per host)
"MaximumTotalConnections"	total maximum http connections
"ConnectionTimeout"	connection timeout
"SocketReadTimeout"	socket layer read timeout