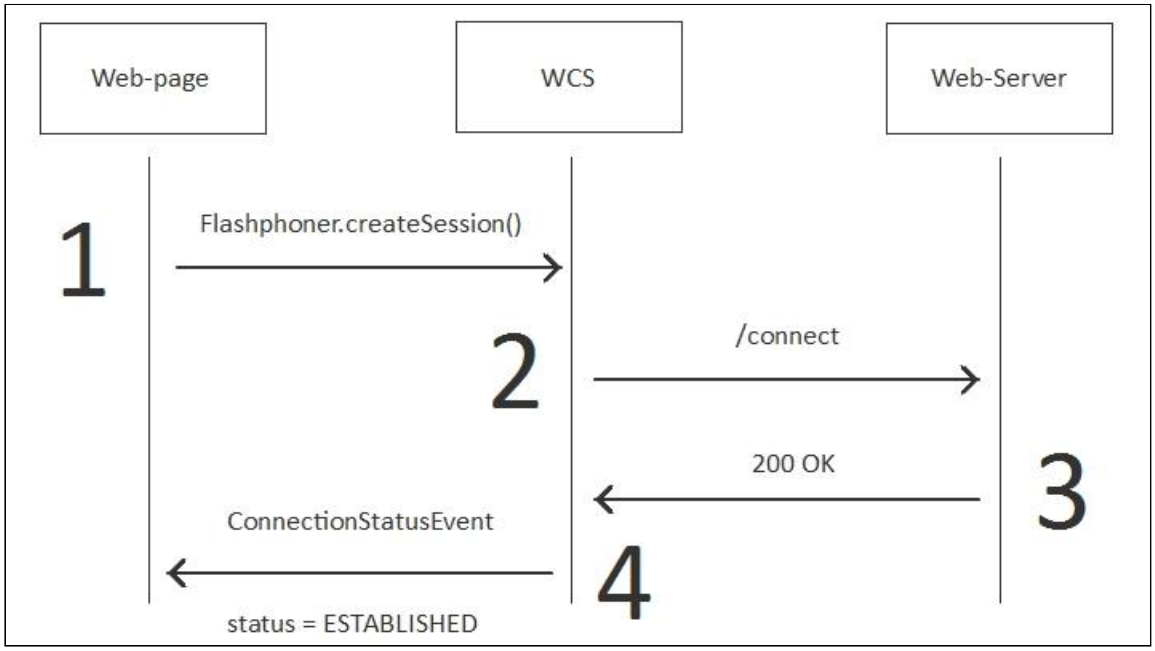
Type 1 - the connect method

Workflow

This method is invoked only once when the browser page connects to the WCS server via the Websocket protocol.

Execution of the connect method is crucial, because this method is responsible for server connection authentication, and if it won't run or stops with an error, the WCS server will decline the connection attempt.



1. On the browser side, Flashphoner.createSession() is executed. After that the browser attempts to establish connection to WCS
2. WCS calls the `connect` REST method
3. WCS receives `200 OK` response and based on the answer, authorizes this connection attempt
4. WCS sends a confirmation to the browser as a `ConnectionStatusEvent` event with the `ESTABLISHED` status.

Example:

REST hook

```
POST /EchoApp/connect HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_45
Host: localhost:8081
Connection: keep-alive
Content-Length: 537

{
   "nodeId":"H4gfHeULtX6ddGGUWwZxhUNyqZHUFH8j@192.168.1.59",
   "appKey":"defaultApp",
   "sessionId":"/192.168.1.38:64604/192.168.1.59:8443",
   "useWsTunnel":false,
   "useWsTunnelPacketization2":false,
   "useBase64BinaryEncoding":false,
   "mediaProviders":[
      "WebRTC",
      "WSPlayer"
   ],
   "clientVersion":"0.5.16",
   "clientOSVersion":"5.0 (Windows NT 6.3; WOW64)",
   "clientBrowserVersion":"Mozilla/5.0 (Windows NT 6.3; WOW64)"
}
```

Backend response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 26 Feb 2017 23:54:06 GMT


{
   "nodeId":"H4gfHeULtX6ddGGUWwZxhUNyqZHUFH8j@192.168.1.59",
   "appKey":"defaultApp",
   "sessionId":"/192.168.1.38:64604/192.168.1.59:8443",
   "useWsTunnel":false,
   "useWsTunnelPacketization2":false,
   "useBase64BinaryEncoding":false,
   "mediaProviders":[
      "WebRTC",
      "WSPlayer"
   ],
   "clientVersion":"0.5.16",
   "clientOSVersion":"5.0 (Windows NT 6.3; WOW64)",
   "clientBrowserVersion":"Mozilla/5.0 (Windows NT 6.3; WOW64)"
}
```

Backend response implementation requirements

In `200 OK` response to `/connect`, backend server must return all of the fileds received from WCS (mirror response). Also, a new fields can be added to backend response, `restClientConfig` for example (see below)

```
{
   "nodeId":"H4gfHeULtX6ddGGUWwZxhUNyqZHUFH8j@192.168.1.59",
   "appKey":"defaultApp",
   "sessionId":"/192.168.1.38:64604/192.168.1.59:8443",
   "useWsTunnel":false,
   "useWsTunnelPacketization2":false,
   "useBase64BinaryEncoding":false,
   "mediaProviders":[
      "WebRTC",
      "WSPlayer"
   ],
   "clientVersion":"0.5.16",
   "clientOSVersion":"5.0 (Windows NT 6.3; WOW64)",
   "clientBrowserVersion":"Mozilla/5.0 (Windows NT 6.3; WOW64)",
   "restClientConfig": {
      ...
   }
}
```

Empty response is also possible, with `Content-Lengh: 0`

```
HTTP/1.1 200 OK
Date: Fri, 20 Nov 2020 03:23:57 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16
X-Powered-By: PHP/5.4.16
Content-Type: application/json
Content-Length: 0
```

A field valuse can be modified if necessary. In this case, the following fields must not be modified or removed:

```
  ...
  "clientVersion" : "0.5.28",
  "clientOSVersion" : "5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/87.0.4280.66 Safari/537.36",
  "clientBrowserVersion" : "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.66 Safari/537.36",
  ...
```

For example, `clientBrowserVersion` field is used later to detect DTLS version supported by browser while establishing WebRTC connection. If there is no this field in response body, or the field is empty, DTLS 1.0 will be used, and WebRTC will not work in latest Chrome and Safari builds.

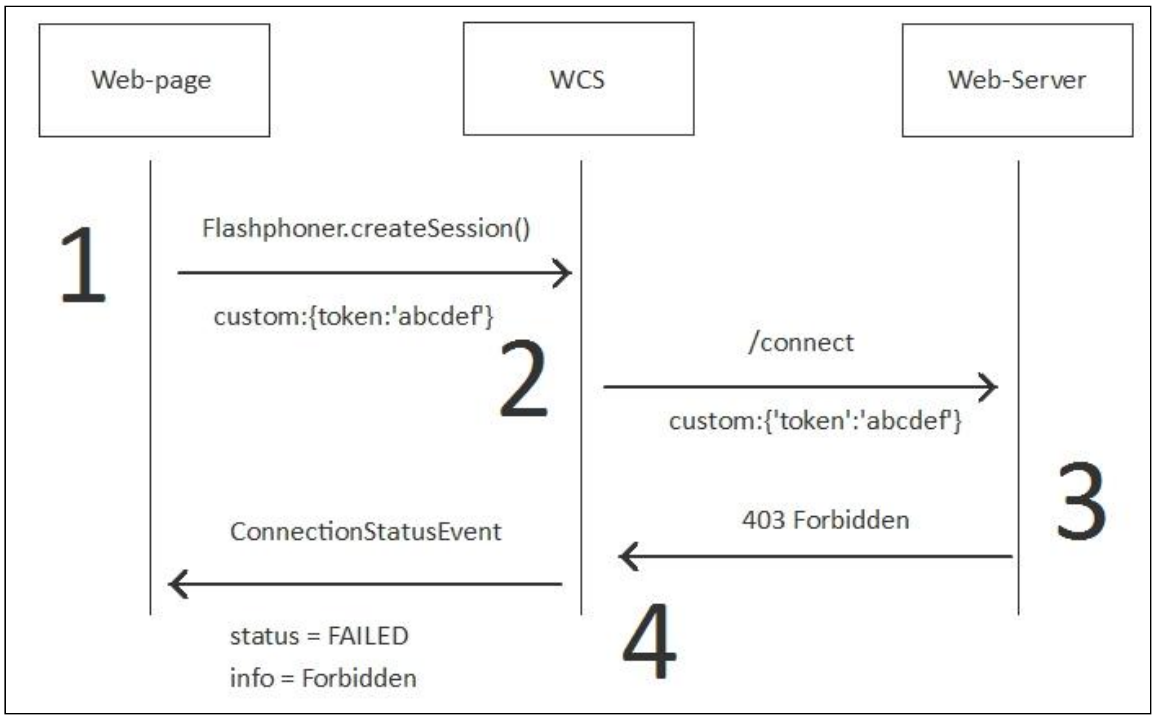
Authentication

By default, WCS successfully authenticates all Websocket connections. That is, the built-in backend server at `http://localhost:8081/apps/EchoApp/connect` always returns `200 OK`.

You can redefine this behavior and make your web server return `403 Forbidden`, in this case WCS will abort the incoming connection.

You can send a token, a password or any other information from the web page using the `custom` field.

WCS transfers this field to the web server as the JSON body, and the backend server can decide whether or not to authenticate based on these data.



Example:

REST hook

```
POST /rest/my_api/connect HTTP/1.1
Accept: application/json
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_111
Host: 192.168.1.101
Connection: keep-alive
Content-Length: 578

{
   "nodeId":"Hw47CFMBEchVOpBMDr29IxjudnJ1sjOY@192.168.1.101",
    "appKey":"defaultApp",
    "sessionId":"/192.168.1.102:60204/192.168.1.101:8443",
    "useWsTunnel":false,
    "useWsTunnelPacketization2":false,
    "useBase64BinaryEncoding":false,
    "mediaProviders":[
        "WebRTC",
        "WSPlayer"
    ],
```

```
    "clientVersion":"0.5.16",
    "clientOSVersion":"5.0 (Windows NT 6.3; Win64; x64)",
    "clientBrowserVersion":"Mozilla/5.0 (Windows NT 6.3; Win64; x64)",
    "custom":{
        "token":"abcdef"
    }
}
```

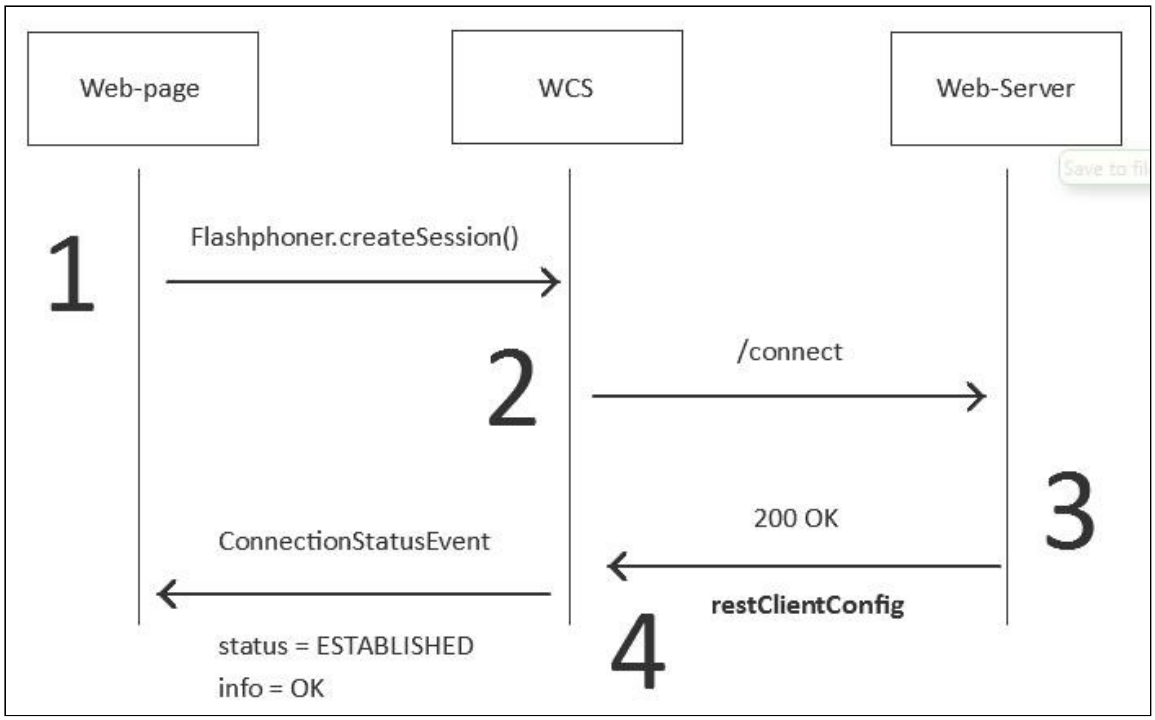
Backend response

```
HTTP/1.1 403 Forbidden
Date: Tue, 28 Feb 2017 09:05:56 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

Configuring other REST methods

During authentication you can configure how should operate all other REST methods invoked after the connection is established, for example: `playStream`, `publishStream`, `StreamStatusEvent`, and so on. To do this, the web server should return the restClientConfig field in the JSON body of the `200 OK` response.

`restClientConfig` is a JSON object that can hold a config to any existing REST method except `connect`.



Example of the JSON body of the `200 OK` response with the `restClientConfig` field.

```
{
  "nodeId" : "kPeSvMn1PFqIMCxZ1Ry6dJ0JNiZ9cqZw@89.179.119.95",
  "appKey" : "defaultApp",
  "sessionId" : "/172.16.16.139:49405/89.179.119.95:8443",
  "useWsTunnel" : false,
  "useWsTunnelPacketization2" : false,
  "useBase64BinaryEncoding" : false,
  "mediaProviders" : [ "WebRTC", "WSPlayer" ],
  "clientVersion" : "0.5.12",
  "clientOSVersion" : "5.0 (Android 5.1.1)",
  "clientBrowserVersion" : "Mozilla/5.0 (Android 5.1.1; Mobile; rv:50.0)
Gecko/50.0 Firefox/50.0",
  "restClientConfig":
  {
     "publishStream":
     {
        "restExclude":"",
        "clientExclude":"",
        "restOnError":"FAIL",
        "restPolicy":"NOTIFY",
        "restOverwrite":""
     }
  }
}
```

This response is constructed according to the following rules:

1. We return data received from WCS unchanged.
2. Add the field:

   ```
   "restClientConfig":{
      "publishStream":{
         "restExclude":"",
         "clientExclude":"",
         "restOnError":"FAIL",
         "restPolicy":"NOTIFY",
         "restOverwrite":""
      }
   }
   ```

This configuration tells the WCS server that invocation of the `publishStream` REST method will comply with the following rules:

1. `restExclude` is empty.
   This means all fields in the JSON body of the response such as `name`, `width`, `height` and so on will be sent from WCS to the web server and none of these fields will be excluded from this query.

2. `clientExclude` is empty.
   Not used in the `publishStream` REST method.
3. `restOnError: FAIL`
   This means that if an error occurred while sending the query to the backend server or if the web server returned 4xx HTTP status, the WCS server should forbid and interrupt publishing the stream.
   The default value is `restOnError: LOG`. This means, WCS simply logs all occurring errors, do not forbid or interrupt operations such as `publishStream`, even if the web server returns `403 Forbidden`.
4. `restPolicy: NOTIFY`
   This means if the backend server returns data different from those it received from WCS, the data are not applied. For instance, if WCS sent the field `name: "stream1"`, and the backend server returned `name="stream2"` in the body of the `200 OK` response, the new value **will not** be applied, because of the `NOTIFY` policy.
5. `restOverwrite` is empty.
   This setting is only applied if the previous `restPolicy` is set to `OVERWRITE`, that is it does not allow redefining fields with data received in the JSON body of the `200 OK` response. The current value is empty, which means no fields can be redefined. To redefine a field, the list of fields must be specified explicitly, for example: `restOverwrite="name,width"`

Example:

REST hook

```
POST /rest/my_api/connect HTTP/1.1
Accept: application/json, application
Content-Type: application/json;charset=UTF-8
User-Agent: Java/1.8.0_111
Host: 192.168.1.101
Connection: keep-alive
Content-Length: 550

{
   "nodeId":"Hw47CFMBEchVOpBMDr29IxjudnJ1sjOY@192.168.1.101",
   "appKey":"defaultApp",
   "sessionId":"/192.168.1.102:26518/192.168.1.101:8443",
   "useWsTunnel":false,
   "useWsTunnelPacketization2":false,
   "useBase64BinaryEncoding":false,
   "mediaProviders":[
       "WebRTC",
       "WSPlayer"
   ],
   "clientVersion":"0.5.16",
   "clientOSVersion":"5.0 (Windows NT 6.3; Win64; x64)",
   "clientBrowserVersion":"Mozilla/5.0 (Windows NT 6.3; Win64; x64)"
}
```

Backend response

```
HTTP/1.1 200 OK
Date: Tue, 28 Feb 2017 10:11:03 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 833
Connection: close
Content-Type: application/json

{
  "nodeId":"Hw47CFMBEchVOpBMDr29IxjudnJ1sjOY@192.168.1.101",
   "appKey":"defaultApp",
   "sessionId":"\/192.168.1.102:26518\/192.168.1.101:8443",
   "useWsTunnel":false,
   "useWsTunnelPacketization2":false,
   "useBase64BinaryEncoding":false,
   "mediaProviders":[
       "WebRTC",
       "WSPlayer"
   ],
   "clientVersion":"0.5.16",
   "clientOSVersion":"5.0 (Windows NT 6.3; Win64; x64)",
   "clientBrowserVersion":"Mozilla\/5.0 (Windows NT 6.3; Win64; x64)",
   "restClientConfig":{
       "publishStream":{
           "clientExclude":"",
           "restExclude":"recordName",
           "restOnError":"FAIL",
           "restPolicy":"NOTIFY",
           "restOverwrite":""
       },
       "playStream":{
           "clientExclude":"",
           "restExclude":"",
           "restOnError":"LOG",
           "restPolicy":"OVERWRITE",
           "restOverwrite":"height,width"
       }
   }
}
```