D2Api

The D2Api interface represents a basic communication channel with the D2000 system for web applications. This communication interface is implemented on the client's side in JavaScript and is built on the CometD library.

Establishing and Disconnecting Connection with D2000

Establishing a connection with D2000 is realized by creating an instance of the *D2Api* class to which the CometD configuration will be submitted and by the following calling the *connect* method. Parameters of the configuration object for CometD are described in the documentation of the CometD configuration.

The only mandatory attribute is the *url* attribute that defines a path to the CometD servlet on the server - it may be entered as an absolute path ("http://server:port/aplikacia/api/cometd") or as a relative path within the web server ("/application/api/cometd").

To other useful attributes, there belongs especially the *maxNetworkDelay* attribute which defines the period in milliseconds and after the period is up, the request on the server is considered failed. It is good to increase this parameter from the default 10 seconds to a higher number if common synchronously calling RPC procedures have the processing period longer.

Optionally, it is possible to define the operation of error states by the *registerErrorHandler* method - within the SmartWeb Framework, the *reportError* functio n serves for this purpose and it displays located dialogue in a case of an error.

```
import D2Api from "framework\d2\d2Api"
import reportError from "framework\basic\reportError"


let cometdConfiguration = {
    url: "/smartWeb/api/cometd",
    maxNetworkDelay: 60000
}
let d2Api = new D2Api(cometdConfiguration);
d2Api.registerErrorHandler(reportError);
d2Api.connect();
```

For disconnecting a connection with D2000 serves the *disconnect* method.

```
d2Api.disconnect();
```

Listening for Value Changes of D2000 Objects

Registering for the subscription of values of D2000 objects is possible using the *subscribeObject* method. The name of the D2000 object, changes of which will be monitored and the service function, which will be called when the value of the object is changed, will be passed through parameters to the method. Optionally, as the third parameter, an object with attributes *returnFields* or *returnTransformation* - that will define which attributes of value should be returned or else the transformation that should be done on values - can be passed. Possible values of *returnFields* and *returnTransformation* attributes are described in the section Serialization of Data between Client and API Interface.

```
// Service function of change of object value
function onSecChange(subscription) {
    // object subscription.uniVal will contain value and attributes of D2000 object
...
// Registration of value subscription
d2Api.subscribeObject("Sec", onSecChange, {returnFields: ["FormattedValue"]});
```

Subscription of object values can be ended by calling the *unSubscribeObject* method where there will be a registered service procedure or a component of the SmartWeb Framework passed as a parameter.

```
d2Api.unSubscribeObject(onSecChange);
```

Reading Values of D2000 Archive Objects

Reading values of D2000 archive objects is possible by the *loadArchive* and *loadArchives* method that read values of one or more archive objects in the given interval. Functions return the Promise object which, in the case of success, will contain completely read data. For calling these functions, it is possible to use also the async/await syntax.

```
var extParameters = {};
var receiveDataStreamHandler = null;


// Reads values of H.ArchivedValue object in interval <startDate, endDate>
var arcData = await d2Api.loadArchive("H.ArchivedValue", startDate, endDate, extParameters,
receiveDataStreamHandler).call();

// reads values of H.ArchivedValue1 and H.ArchivedValue2 objects in interval <startDate, endDate>
var arcDataMulti = await d2Api.loadArchives(["H.ArchivedValue1", "H.ArchivedValue2"], startDate, endDate,
extParameters).call();
```

The optional parameter *extParameters* serves for defining attributes of values which should be returned - the list is defined in the *returnFields* attribute - described in the section Serialization of Data between Client and API Interface. In addition to that, oversampling of values using the *oversampleSeconds* pa rameter (the step of oversampling is in seconds) can be defined by the *extParameters* parameter and also it is possible to restrict the number of returning values from an archive by setting the *limitDataLength* attribute. In a case of omitting the *extParameters* parameter, only values with no other attributes will be returned and no data oversampling nor data trimming will occur.

```
var extParameters = {
    returnFields: ["ValueTime"], // besides values, time stamps of values are required
    oversampleSeconds: 3600,     // values were oversampled with hour step
    limitDataLength: 1000        // maximally, there will be 1000 values returned
}
```

When reading more data from an archive, the data from the D2000 system are continuously streamed in batches. If it is desired to collect and process those partial data, it will be possible using optional parameter *receiveDataStreamHandler* which defines a function with two parameters - function with two parameters - in the first one, there will be values of a batch submitted and in the second one, there will be an indication whether it is the last batch.

receiveDataStreamHandler

```
function receiveDataStreamHandler(data, noModeData) {
    // data processing
}
```

Calling D2000 RPC

Same as the REST API, D2Api also enables calling D2000 RPC procedures written in ESL or in Java. The following methods serve for it: *rpc* - calling ESL RPC, *rpcJava* - calling Java RPC and *rpcSBA* - calling Java RPC for transmission of binary data. The first two parameters of these methods define the name of the object event (script) and the name of the RPC procedure. All other parameters are submitted as parameters into the calling RPC procedure. Structure of parameters for calling RPC is described in the section Serialization of Data between Client and API Interface. Methods return an object with the *call* method which does the RPC calling. Returning value of RPC calling is the Promise object which, in the case of a successful calling, contains required output parameters of the RPC procedure as its attributes in the form of the Unival value.

```
// Calls RPC procedure Sum on Event script E.SmartWebTutorial
const rpcResponse = await d2Api.rpc(
    "E.SmartWebTutorial",
    "Sum",
    1, // first parameter - input
    2, // second parameter - input
    {type: "real", returnAs: "sum"} // third parameter - output
).call();
// rpcResponse.sum contains output parameter RPC as UniVal value
```

Called RPC procedure in ESL has the following prescript:

```
RPC PROCEDURE Sum (IN REAL _a, IN REAL _b, REAL _c)
  _c := _a + _b
END Sum
```

Calling JavaScript Functions from D2000

D2Api enables also reverse calling from D2000 to a web browser in a form of registered JavaScript functions. Such callings are useful mainly for notifications of various events which emerge asynchronously in the D2000 system. For registering the JavaScript function which can be called from D2000 serves the *subscribeRpc* method. Its parameters are the name of the RPC procedure, under which the function will be available within the D2000 system, and the JavaScript function itself which will be called in return calling. All parameters of the function are objects of the Unival type. For deregistering of the function of return calling serves the *unSubscribeRpc* method with one parameter - the given JavaScript function.

```
// Registered Javascript function
smartWebSessionRPC(message) {
    // ...
}

// Registration of function
d2Api.subscribeRpc("SmartWebSessionRPC", smartWebSessionRPC);

// Deregistration of function
d2Api.subscribeRpc(smartWebSessionRPC);
```

So the function can be called from D2000, the session, on which it should be called, must be known. This can be found out by calling arbitrary initializing RPC procedure in which the process, from which it was called, is identified.

```
INT _caller ; HOBJ SmartWeb session which requested return callings

RPC PROCEDURE InitCallbacks
  _caller := %GetRPCCallerProcess()
END InitCallbacks
```

Then, calling of registered JavaScript function from ESL looks like this:

```
CALL [(0)] SmartWebSessionRPC("Hello SmartWeb") ASYNC ON (_caller)
```

Because one SmartWeb session can have under one name of the RPC procedure registered more JavaScript functions, it is a multiple calling (multicast) in which output parameters cannot be used. Calling must be asynchronous at the same time.

Password Change of Logged in User

The D2Api interface also enables to change the password of a logged in user. The *changePassword* method serves for that; it has two parameters - an old password, a new password and it returns the Promise object.

```
let oldPassword = 'secret';
let newPassword = 's3Cr3t*';
d2Api.changePassword(oldPassword, newPassword).
    .then(onFulfilled => {
        if (onFulfilled.data === 'ok') {
            // password changed
        }
    });
```