Webpanel API

Communication is based on HTTP protocol. Clients send HTTP POST request and get the response depending on the request. Each request must specify the credentials in case the credentials are specified for the project.

Login request

Data:

```
{
      ver : 1,//version
      //optional - only for non-authenticated access
      //credentials
      c: {u: "karel", p: "x"}

}
```

Login response

```
{
      ver : 1,//version
      code: "ok", //return code
      msg: "", //error message
      c : {u: "karel", p:"x"} //must be used for further
      communication
}
```

Data request

Data:

```
{
      ver : 1,//version
      //optional - only for non-authenticated access
      //credentials
      c: {u: "karel", p: "x"},
      //variable IDs
      v: [ "svc://defaultConnection/myVarId1",
           "svc://otherConnection/xxxYYYYDDD"]
}
```

- variable ID is unique string - in future, may be different for different platforms/protocol versions (uPLC - int?, SharkRT - string)

Variable specification for RT2

- svc - protocol

- Connection - generally, clients can request data from more sources (more or less corresponds to a connection definition of PLC). For local data (data belonging to the PLC that executes the request) it may be defaultconnection.
- Variable id: it has a form of commUID[offset, lenght]. CommUID can be retrieved from a .vlist file of the IDE 2 project and specifies a variable. Variables can be of a complex and large types. In this case a user can retrieve only a small part of the variable by using the offset and length.
- Example: svc://defaultConnection/8776[0,2]
    - Represents a variable with the commUID 8776. 2 bytes from the offset 0 are returned.

the POST body can be:

1) JSON string (AngularJS default behaviour):
{"ver":1,"c":{"u":"karel","p":"x"},"v":["svc://defaultConnection/myVarId1","svc://otherConnection/xxxYYYYDDD"]}

2) converted into POST parameters (jQuery default behaviour):

ver=1&c%5Bu%5D=karel&c%5Bp%5D=x&v%5B%5D=svc%3A%2F%2FdefaultConnection%2FmyVarId1&v%5B%5D=svc%3A%2F%2FotherConnection%2FxxxYYYYDDD

The best solution would be to support both versions. Easier is now the JSON string.

Data response

```
{
      ver : 1,//version
      code: “ok”, //return code
      msg: “”, //error message
      v: [{
           i: “svc://defaultConnection/myVarId1”, //variable Id
           v: ”ACA1”, //value - binary encoded
           q: “G”,//quality
           t: 654654321654 //timestamp UTC, milliseconds since 1970
           }
        ]
}
```

Encoding of values - version 1

- Each value is transmitted as a big endian binary representation of the specified variable (and its offset and length).

Command Request

```
{
      ver : 1,//version
     //optional - only for non-authenticated access
```

```
      //credentials
      c: {u: "karel", p: "x"},
      v: [{
           i: "svc://defaultConnection/myVarId1", //variable Id
           //digital setter
           time: "20.0", //digital setter has some time property...
           state: "2", //state that should be applied
           defaultValue: "6", //some default value
           //analogPlusMinusInit, TPG, ...
           set: 5,
           //set: "<?cdata....?>" //TPG
           //login - pin
           login: "1234"
           }
         ]
}
```

```
uPLC Warning : only one variable is expected in the request - other variables are ignored
```

Historical Data Request

```
{
      ver : 1,//version
     //optional - only for non-authenticated access
      //credentials
      c: {u: "karel", p: "x"},
      v: [ {i:"svc://defaultConnection/myVarId1",
            from: XXXX,
            to: YYYY
            },
           {i:"svc://defaultConnection/myVarId1",
            from: XXXX,
            to: YYYY
           }
         ]
}
```

Configuration Response

```
{
      ver : 1,//version
      platform: "uPLC",
      maxVariablesInRequest: 5,
      maxRequestSize: 1500, //http body size in bytes
```

```
    preferedLanguage: “en-US”
}
```

Native/built-in data types

Note: big endian encoding (v1)

- integer (represents int8-int64)
- unsigned integer (represents uint8-uint64)
- double (double64)
- float (float32)
- bool (boolean - 0 ~ false, 1 ~ true)
- dt (date time - the number of 100-nanosecond intervals that have elapsed since 12:00:00 midnight, January 1, 0001)
- time (time - the number of 100-nanosecond intervals)
- Enums are encoded into 32-bit int