

# D2000 REST API

REST API rozhranie implementované SmartWeb platformou môžeme rozdeliť na nasledovné asti:

- rozhranie na autentifikáciu
- rozhranie na prístup k dátam a službám D2000 systému
- administrátorské rozhranie na monitorovanie volaní do D2000 a stavu SmartWeb servera

Tieto oblasti REST API rozhrania sú popísané v nasledujúcich kapitolách.

- [Autentifikácia](#)
- [Naíťavanie hodnôt z archívu](#)
- [Naíťavanie hodnôt EDA vektora](#)
- [Zápis hodnôt do EDA vektora](#)
- [Volanie D2000 RPC metód](#)
- [Volanie D2000 SBA RPC metód](#)
  - [Stiahnutie binárnych dát z D2000 cez HTTP GET \(URL linku\)](#)
  - [Posielanie binárnych dát do D2000 cez HTTP POST](#)
    - [Odporúčaný spôsob kódovania vstupných parametrov spolu s binárnym obsahom](#)
    - [Automatické kódovanie vstupných parametrov spolu s binárnym obsahom](#)

## Autentifikácia

Podporované spôsoby autentifikácie pre REST API sú [HTTP-BASIC](#) a [Api Keys](#).

Pre HTTP-BASIC typ autentifikácie sa posiela používateľské meno a heslo priamo v hlavičke každej HTTP požiadavky. To znamená, že každý REST API request automaticky aj autentifikuje používateľa. V prípade neúspešnej autentifikácie server vráti v hlavičke odpovede HTTP status 404. Z tohto dôvodu nie je potrebné mať explicitné prihlasovanie do REST API rozhrania cez špeciálnu URL. Napriek tomu je optimálne tú funkciu extrahovať, kvôli aplikáciám v ktorých sa používatelia explicitne prihlasujú a teda aplikácia potrebuje overiť zadané meno a heslo.

Pre Api Keys typ autentifikácie sa v HTTP hlavičke "X-API-Key" posiela aplikovaný ku - vygenerovaný náhodný reazec, prostredníctvom ktorého je možné sa autentifikovať. Výhodou tohto spôsobu autentifikácie je, že sa neposiela meno a heslo do D2000 priamo v requeste. Smart Web aplikácia má tento reazec vo svojej konfigurácii spárovaný s používateľským menom a heslom do D2000. Prihlasovacie údaje ako aj aplikovaný ku sú bezpečne uložené v špeciálnom keystore na disku, veda konfigurovaného súboru smartweb.json. Správu týchto kuov je možné editovať cez Smart Web admin konzolu (web aplikáciu) dostupnú na URL: <https://<doména.sk>/<názov aplikácie>/admin>.

Overenie úspešnosti autentifikácie je možné spraviť aj samostatne odoslaním prázdnej GET požiadavky s HTTP-BASIC alebo Api Keys autentifikáciou na adresu:

GET <https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/auth/login>

Taktiež odhlásenie je možné realizovať odoslaním prázdnej GET požiadavky s HTTP-BASIC autentifikáciou na adresu:

GET <https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/auth/logout>



Volanie explicitného prihlásenia a odhlásenia nie je nevyhnutné, pretože autentifikované údaje (HTTP BASIC alebo Api Keys) sa aj tak posielajú s každou požiadavkou v HTTP hlavičkách. A odhlásenie prebehne aj automaticky v prípade automatickej expirácie session do D2000 konfigurovanej v [autentifikanej asti konfigurácie](#) SmartWeb Platformy. Niekedy je ale výhodné používať explicitné prihlásenie a odhlásenie v prípadoch, keď toto API je využívané napr. z mobilnej aplikácie veľkým počtom používateľov.

## Naíťavanie hodnôt z archívu

Naíťavanie hodnôt z archívu je možné cez GET požiadavku s HTTP hlavičkou Content-Type: application/json na adresu:

GET <https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/archive/<meno archívneho objektu>?beginTime=<celé číslo>&endTime=<celé číslo>&oversampleSeconds=<celé číslo>&limitDataLength=<celé číslo>returnFields=<text>>

Význam jednotlivých parametrov je nasledovný:

| Meno parametra    | Typ                                    | Povinný | Popis                                                                                           |
|-------------------|----------------------------------------|---------|-------------------------------------------------------------------------------------------------|
| beginTime         | celé číslo (poet milisekúnd od epochy) | áno     | začiatok časového intervalu vyžiadaných hodnôt archívu                                          |
| endTime           | celé číslo (poet milisekúnd od epochy) | áno     | koniec časového intervalu vyžiadaných hodnôt archívu                                            |
| oversampleSeconds | celé číslo (poet sekúnd)               | nie     | dĺžka intervalu v sekundách pre oversampling dát, ak nie je definovaný vráti sa originálne dáta |
| limitDataLength   | celé číslo (poet hodnôt)               | nie     | maximálny počet vrátených hodnôt, ak nie je definovaný vráti sa všetky hodnoty z intervalu      |

|              |                                                    |     |                                                                                                       |
|--------------|----------------------------------------------------|-----|-------------------------------------------------------------------------------------------------------|
| returnFields | text (definované Unival atribúty oddelené iarkou ) | nie | vyžiadané Unival atribúty, ktoré budú vrátené spolu s asovou známkou a hodnotou. Napr: "Status,Flags" |
|--------------|----------------------------------------------------|-----|-------------------------------------------------------------------------------------------------------|

Například pre nasledujúce HTTP GET volanie:

GET <http://localhost/smartWeb/api/rest/v0/d2/archive/H.AdeunisRF1External?beginTime=504501912000&endTime=1504601912000>

dostaneme zo servera odpove s poom archívnych hodnôt:

```
[
  [
    1503492475090,
    23.2
  ],
  [
    1503642560209,
    23.2
  ],
  [
    1503643165774,
    23.3
  ],
  ...
]
```

Každá archívna hodnota je reprezentovaná kvôli vekosti prenášanej správy samostatným poom, priom prvý prvok poa je vždy asová známka a druhý samotná hodnota. V prípade definovania alších návratových polí parametrom returnFields sú tieto parametre vrátené v alších prvkoch poa poda poradia ako boli definované.

## Naítavanie hodnôt EDA vektora

V prípade ak je nakonfigurované spojenie na EDA server, dá sa realizova naitavanie a zápis hodnôt do EDA vektorov.

Naítavanie hodnôt EDA vektora je možné cez GET požiadavku s HTTP hlavičkou Content-Type: application/json na adresu:

GET <https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/eda/<kód eda vektora>?beginTime=<celé číslo>&endTime=<celé číslo>&period=<text>>

Význam jednotlivých parametrov je nasledovný:

| Meno parametra | Typ                                                      | Povinný | Popis                                                                                           |
|----------------|----------------------------------------------------------|---------|-------------------------------------------------------------------------------------------------|
| beginTime      | celé číslo (poet milisekúnd od epochy)                   | áno     | zaiatok asového intervalu vyžiadaných hodnôt archívu                                            |
| endTime        | celé číslo (poet milisekúnd od epochy)                   | áno     | koniec asového intervalu vyžiadaných hodnôt archívu                                             |
| period         | reazec alebo číslo identifikujúci prevzorkovaciu periódu | nie     | džka intervalu v sekundách pre oversampling dát, ak nie je definovaný vrátia sa originálne dáta |

Například pre nasledujúce HTTP GET volanie:

GET <http://localhost/smartWeb/api/rest/v0/d2/eda/testvektor?beginTime=1633337999000&endTime=1633339999000>

dostaneme zo servera odpove s poom archívnych hodnôt:

```

{
  "errorCode": "SUCCESS",
  "profilingInfo": {
    "numDbTaskRequests": 1,
    "numDbAccesses": 3
  },
  "values": [
    {
      "status": [
        "Valid"
      ],
      "value": 999.9,
      "time": 1633338999000
    },
    {
      "status": [
        "Valid"
      ],
      "value": 100.1,
      "time": 1633339000000
    }
  ]
}

```

Atribút `errorCode` identifikuje stav odpovede z EDA servera. Možné hodnoty sú `SUCCESS` alebo chybové stavy EDA servera: `ERR_INTERNAL_ERROR`, `ERR_INVALID_PARAM_TYPE`, `ERR_TIMESTEP`, `ERR_INVALID_VALUE_TYPE`, `ERR_DB_ERROR`, `ERR_VECTOR_NOT_EXIST`, `ERR_INVALID_VECTOR_TYPE`, `ERR_LOGON`, `ERR_VECTOR_NAME`, `ERR_CONVERT_ERROR`, `ERR_NOT_IMPLEMENTED`, `ERR_RANGE_ERROR`, `ERR_VALUE_STATUS`, `ERR_END_WITHOUT_RETURN`, `ERR_SYNTAX_ERROR`, `ERR_EMPTY_SET_OF_VECTORS`, `ERR_CACHE_NOT_FOUND`, `ERR_CACHE_READ_CHANGED`, `ERR_CACHE_TOO_MANY_READS`, `ERR_CACHE_READ_WRITE_MISMATCH`, `ERR_CACHE_RECURSIVE_VECTOR_DEFINITION`, `ERR_CACHE_NO_MEMORY`, `ERR_TBLSpace_NOT_EXIST`, `ERR_TIMESTEP_FOR_TBLSpace`, `ERR_TBLSpace_ALREADY_EXIST`, `ERR_INSUFFICIENT_RIGHTS`, `ERR_CANCELED`, `ERR_VERSION_DOESNT_EXIST`, `ERR_NOT_ARCHIVED`, `ERR_ARCHIVE_ERROR`, `ERR_VECTOR_ALREADY_EXISTS`, `ERR_VERSION_ALREADY_EXISTS`, `ERR_ENV_DOESNT_EXIST`, `ERR_BATCH_DOESNT_EXIST`, `ERR_BATCH_ALREADY_RUNNING`, `ERR_BATCH_TOO_MANY_BATCHES`, `ERR_OUT_OF_MEMORY`, `ERR_PARAM_BLOCK_DOESNT_EXIST`, `ERR_PARAM_BLOCK_TOO_MANY`, `ERR_VALIDATION_FAILED`.

Atribút `profilingInfo` uvádza profilovacie informácie o obsluhu požiadavky. V atribúte `values` sa nachádzajú hodnoty EDA vektora za daný interval.

## Zápis hodnôt do EDA vektora

Zápis hodnôt EDA vektora je možné realizovať cez POST požiadavku s HTTP hlavičkou `Content-Type: application/json` na adresu:

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/eda/<kód eda vektora>`

Napríklad pre nasledujúce HTTP POST volanie s nasledujúcim obsahom POST requestu, zapíšeme hodnoty do EDA vektora:

POST <http://localhost/smartWeb/api/rest/v0/d2/eda/testvektor>

### obsah POST requestu

```

{
  "values": [
    {
      "value": 999.9,
      "time": 1633338999000,
      "status": ["Valid"]
    },
    ... <dalsie hodnoty>
  ]
}

```

odpoveď na zápis môže byť nasledovná

```
{
    "errorCode": "SUCCESS",
    "profilingInfo": {
        "numDbTaskRequests": 2,
        "numDbAccesses": 12
    }
}
```

Atribút `errorCode` identifikuje stav odpovede z EDA servera. Možné hodnoty sú uvedené v predchádzajúcej podkapitole.

## Volanie D2000 RPC metód

Cez REST rozhranie je možné vola D2000 RPC procedúry napísané v ESL aj v Java. Volanie ESL RPC procedúr prebieha odoslaním POST požiadavky s HTTP hlavičkou `Content-Type: application/json` na jednu z adries:

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/rpc/<meno eventu>/<meno RPC metódy>`

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/rpc/interface/<meno eventu>/<meno interfacu>/<meno RPC metódy>`

v prípade volania Java RPC su URL nasledovné:

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/rpc/java/<meno eventu>/<meno RPC metódy>`

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/rpc/java/interface/<meno eventu>/<meno interfacu>/<meno RPC metódy>`

Ak chceme volat RPC metodu smeno procesu namiesto mena eventu je možné použiť nasledovné URL adresy:

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/rpc/japi/<meno procesu>/<meno RPC metódy>`

POST `https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/rpc/japi/interface/<meno procesu>/<meno interfacu>/<meno RPC metódy>`

Telo odosielanej správy je JSON pole s parametrami volanej RPC. Výstupom takejto požiadavky sú hodnoty výstupných parametrov RPC uložené v JSON objekte, ktorého atribúty sú požadované názvy výstupných parametrov definované atribútmi `returnAs`. Detaily serializácie parametrov RPC metód boli popísané v [predchádzajúcej kapitole](#). Príklad volania RPC s názvom `TestInOut` na evente `E.E.SmartWebApiTutorial` s 5 parametrami, pričom prvý, tretí a štvrtý parameter sú vstupno-výstupné a definujú [logický názov pre vrácané hodnoty parametrov](#). Vstupné parametre (druhý a piaty) zároveň využívajú [implicitnú konverziu](#) na Unival objekt z jednoduchých JSON typov.

POST `http://localhost/smartWeb/api/rest/v0/d2/rpc/E.E.SmartWebApiTutorial/TestInOut`

### Príklad volania RPC metódy

```
[
  {
    "type": "bool",
    "value": "vTrue",
    "returnAs": "boolParam" // výstupná hodnota bude pod názvom boolParam
  },
  123,
  {
    "type": "real",
    "value": 10.9,
    "returnAs": "realParam", // výstupná hodnota bude pod názvom realParam
    "returnFields": ["ValueTime", "Status"] // k výstupnej hodnote sú požadované aj atribúty ValueTime a Status
  },
  {
    "type": "time",
    "returnAs": "timeParam" // výstupná hodnota bude pod názvom timeParam
  },
  "hello D2000"
]
```

Kód volanej RPC metódy môže byť napríklad:

```

RPC PROCEDURE TestInOut(BOOL _bool, IN INT _int, REAL _real, TIME _time, IN TEXT _text)
  _bool := !_bool
  _real := _real / 2
  _time := SysTime
END TestInOut

```

Výstup z volania RPC je v JSON objekte, ktorý má atribúty podľa požadovaných názvov výstupných parametrov:

#### Výstup volania RPC metódy

```

{
  "realParam": {
    "type": "real",
    "value": 5.45,
    "status": [
      "valid"
    ],
    "valueTime": 1498213794522
  },
  "boolParam": {
    "type": "bool",
    "value": "vFalse"
  },
  "timeParam": {
    "type": "time",
    "value": 1498213794012
  }
}

```

## Volanie D2000 SBA RPC metód

Pretože D2000 nepozná binárny dátový typ, RPC procedúry nie sú na presun binárnych dát vhodné. Na tento účel slúžia Simple Byte Array (SBA) metódy napísané v D2000 Java.



SBA RPC je v princípe [Java RPC metóda](#), ktorá má jeden vstupný parameter typu pole bajtov (byte[]) a taký istý výstupný parameter.

## Stiahnutie binárnych dát z D2000 cez HTTP GET (URL linku)

Na stiahnutie binárnych dát z D2000 na klienta slúži GET príkaz s HTTP hlavičkou Content-Type: application/octet-stream na adrese:

GET https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/sba/<meno eventu>/<meno SBA RPC metódy>?fileName=file.dat [&parameterX=hodnotaX&parameterY=hodnotaY&...]

Parametre dotazu (as za otáznikom) sú automaticky preposlané do SBA metódy tak, ako boli zadané do adresy. Parameter fileName definuje meno saňovaného súboru pod ktorým webový prehliadač uloží súbor na disk. Ak sa tento parameter nenastaví, tak názov downloadovaného súboru bude implicitne "file.dat". Okrem toho Smart Web server pridáva pre SBA RPC ešte parameter sessionUserName, kvôli identifikácii používateľa ktorý SBA RPC volá. Ostatné parametre závisia od konkrétnej implementácie volania SBA RPC metódy. SBA RPC metóda všetky zadané parametre dostane vo vstupnom poli bajtov (byte[]).

Nasledujúci príklad ilustruje volanie a implementáciu SBA RPC metódy pre stahovanie konkrétneho súboru pdf reportu z lokálneho súborového systému. Pozor uvedený príklad nie je vôbec vhodný pre reálne použitie a je uvedený iba pre ilustráciu použitia volania SBA RPC. Volanie SBA RPC metódy reportContract\_PDF v evente E.E.SmartWebApiTutorial s parametrom id ktorý identifikuje číslo reportu a tým pádom saňovaného súboru.

GET http://localhost/smartweb/api/rest/v0/d2/sba/E.E.SmartWebApiTutorial/reportContract\_PDF?fileName=report.pdf&id=10

Implementácia SBA RPC metódy je nasledovná:

```

package app.runnables;

import app.wrappers.E$E.SmartWebApiTutorial__$WRAPPER$__;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

public class E$E.SmartWebApiTutorial extends E$E.SmartWebApiTutorial__$WRAPPER$__ {

    public byte[] reportContract_PDF(byte[] urlParamsBytes) throws IOException {
        // naparsovanie poslaných URL parametrov do hash-mapy
        final HashMap<String, String> parameters = getParametersFromUrlQueryString(urlParamsBytes);
        // Vyparsovanie parametra id do premennej
        final long contractId = Long.parseLong(parameters.get("id"));
        // Získanie mena aktuálneho používateľa volajúceho SBA RPC
        final String userName = parameters.get("sessionUserName");
        System.out.println("DEBUG: Downloading contract with id " + contractId);
        // Vyrobenie cesty k súboru s daným reportom
        Path path = Paths.get("D:/D2000/Contract" + contractId + ".pdf");
        // Načítanie obsahu súboru a jeho vrátenie ako pole bajtov
        return Files.readAllBytes(path);
    }

    /**
     * Utility metóda, vráti naparsované URL parametre z vstupného poa byte[]
     */
    private HashMap<String, String> getParametersFromUrlQueryString(byte[] urlQueryStringBytes) throws
    UnsupportedEncodingException {
        final String urlParamsString = new String(urlQueryStringBytes, "UTF-8");
        final List<String> paramPartsList = Arrays.asList(urlParamsString.split("&"));
        final HashMap<String, String> parameters = new HashMap<String, String>();
        for (String paramPartsString : paramPartsList) {
            final String[] paramParts = paramPartsString.split("=");
            final String paramName = paramParts[0];
            final String paramValue = paramParts.length > 1 ? paramParts[1] : null;
            parameters.put(paramName, paramValue);
        }
        return parameters;
    }
};

```

## Posielanie binárnych dát do D2000 cez HTTP POST

Posielanie binárnych dát do D2000 je možné cez HTTP POST metódu na identickú url linku pri sahoaní binárnych dát, s tou istou HTTP hlavičkou Content-Type: application/octet-stream.

POST https://<doména.sk>/<názov aplikácie>/api/rest/v0/d2/sba/<meno eventu>/<meno SBA RPC metódy>

Pri posielaní binárnych dát nie je možné posla špecifické URL parametre priamo do SBA RPC metódy. Hodnota vstupného parametra SBA RPC metódy bude v tomto prípade binárny obsah posielaný v tele POST requestu. Klient ale nie je žiadnym spôsobom obmedzený typom obsahu, ktorý posiela cez HTTP POST request. Jediná požiadavkou je, aby binárny obsah vedela rozkódovať príslušná implementácia SBA RPC metódy, analogicky ako v predchádzajúcom prípade ke sme ilustrovali rozkódovanie URL parametrov špeciálnou utility metódou getParametersFromUrlQueryString(). Ak klient potrebuje posielať s binárnym obsahom aj dodatočné vstupné parametre (napr. identifikujúce tento obsah), je zakódovanie a rozkódovanie takéhoto obsahu v SBA RPC plne v jeho kompetencii. Nasledujúca kapitola obsahuje odporúčaný spôsob takéhoto kódovania.

## Odporúčaný spôsob kódovania vstupných parametrov spolu s binárnym obsahom

V prípade že so samotným binárnym obsahom potrebujeme v rámci volania SBA RPC metódy posielať aj ďalšie vstupné parametre, odporúčame kódovať parametre ako aj priložený obsah do zip streamu. Výhodou takéhoto riešenia je univerzálnosť a jednoduchosť použitia vo väčšine programovacích jazykoch (ZIP kompresia býva väčšinou dobre podporená bu v štandardnej knižnici daného jazyka alebo v nejakej inej voľne dostupnej verzii knižnice).

Uvádžame príklad kompresie viacerých parametrov do jedného ZIP byte streamu v Jave na strane klienta:

```

/**
 * Proposal method how to serialize multiple key-value pairs to byte array
 */
@SuppressWarnings("unused")
private byte[] writeParameters(Map<String, byte[]> params) throws IOException {
    try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {
        try (ZipOutputStream zos = new ZipOutputStream(baos)) {
            for (Map.Entry<String, byte[]> entry : params.entrySet()) {
                final ZipEntry zipEntry = new ZipEntry(entry.getKey());
                zipEntry.setSize(entry.getValue().length);
                zos.putNextEntry(zipEntry);
                zos.write(entry.getValue());
                zos.closeEntry();
            }
        }
        return baos.toByteArray();
    }
}

```

Príklad rozkódovania ZIP byte streamu na strane SBA RPC potom bude:

```

/**
 * Proposal method how to deserialize multiple key-value pairs from byte array
 */
@SuppressWarnings("unused")
private Map<String, byte[]> loadParameters(byte[] inputBytes) throws IOException {
    final Map<String, byte[]> dataParts = new HashMap<String, byte[]>();
    try (ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(inputBytes)) {
        try (ZipInputStream zipInputStream = new ZipInputStream(new ByteArrayInputStream(inputBytes))) {
            ZipEntry zipEntry = zipInputStream.getNextEntry();
            byte[] buffer = new byte[4096];
            while (zipEntry != null) {
                final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
                int len;
                while ((len = zipInputStream.read(buffer)) > 0) {
                    outputStream.write(buffer, 0, len);
                }
                final byte[] data = outputStream.toByteArray();
                dataParts.put(zipEntry.getName(), data);
                zipEntry = zipInputStream.getNextEntry();
            }
        }
    }
    return dataParts;
}

```

## Automatické kódovanie vstupných parametrov spolu s binárnym obsahom

Ako alternatíva k predchádzajúcej možnosti kódovania vstupných parametrov spolu s obsahom na strane klienta je volanie SBA RPC metódy cez HTTP POST s inou hodnotou HTTP hlavičky: Content-Type: multipart/form-data. V tomto prípade sa telo POST volania kóduje podľa [definovaného štandardu na posielanie formulárov aj s binárnymi súborami z prehliadača](#). SmartWeb podporuje aj tento formát na volanie SBA RPC metód. Vstupný parameter pri volaní SBA RPC metódy bude v tomto prípade obsahovať obsah hodnôt jednotlivých polí formulára zazipovaný spôsobom ako bol popísaný v predchádzajúcej kapitole. T.j. na rozkódovanie parametrov je možné využiť už uvedenú Java metódu loadParameters.



Smart Web server v tomto prípade tak isto ako pri stiahnutí binárnych dát z D2000 cez HTTP GET, automaticky pridáva pre SBA RPC aj parameter sessionUserName, kvôli identifikácii používateľa ktorý SBA RPC volá.