C1 Core REST/RPC tunneling

REST requests can be invoked using RPC requests.

1. This mandatory for modules to execute REST requests as they can only use RPC to communicate with C1 Core.
2. It enables REST transactions. I. e. multiple REST calls can be bundled and if one of them fails, all previous REST operations are rolled back.

RPC call structure

The method name looks like this:

```
REST/<HTTP method><space character><REST path without API version>
```

So let's say the HTTP method is `POST` and the REST path accessed on C1 Core is `/api/v1/my/rest/path`. Then the method name will be:

```
REST/POST my/rest/path
```

A payload for the REST request can be passed as first parameter to the method.

Providing principal information

For a lot of calls, C1 Core requires information about the accessed principal. This information is mandatorily passed as second parameter. This second parameter is a `Struct` that looks like this:

```
{
  "sp": "<system provider ID>",
  "sd": "<system distributor ID>",
  "bp": "<business partner ID>"
}
```

Providing user information

For a lot of calls, C1 Core requires information about the user executing the REST call. This information is mandatorily passed as third parameter. This third parameter is a `Struct` that looks like this:

```
{
  "type": <user type>,
  "sp": "<system provider ID>",
```

```
  "sd": "<system distributor ID>",
  "bp": "<business partner ID>",
  "id": "<user ID>"
}
```

Using this, a full method call might look like this:

```
{
  "method": "REST/GET economic_units",
  "params":[
    {},
    {
      "sp": "48109350-1db6-11e9-8e66-2f71a0be4cc5",
      "sd": "65490af0-ef62-11e9-847a-a58a2ea33526",
      "bp": "123b7640-a375-11eb-916a-bdf6422f302f"
    },
    {
      "type": 4,
      "sp": "48109350-1db6-11e9-8e66-2f71a0be4cc5",
      "sd": "65490af0-ef62-11e9-847a-a58a2ea33526",
      "bp": "123b7640-a375-11eb-916a-bdf6422f302f",
      "id": "082ad970-07d0-11eb-9a26-ebb1cff8cdc8"
    }
  ]
}
```

REST transactions

For REST transactions multiple method calls can be passed as an `Array`, for example:

```
[
  {
    "method":"REST/POST business_partners",
    "params":{
      "name":"My business partner",
      "city":null,
      "country":null,
      "email":"my-business-partner@mail.com",
      "phone":null,
      "street":null,
      "streetNumber":null,
      "zip":null,
      "password":"my-secret",
      "logo":"my-logo",
      "color":"#e6f2ff",
      "backgroundColor":"#ffe6e6",
      "subdomain":"my-business-partner"
    }
  },
  {
    "method":"REST/POST home_client_groups",
    "params":{
      "name":"Administrator",
```

```
      "acl":{
        "version":1,
        "moduleAccess":"*"
      }
    },
    "sp":"48109350-1db6-11e9-8e66-2f71a0be4cc5",
    "sd":"082ad970-07d0-11eb-9a26-ebb1cff8cdc8",
    "bp":"{{{0.result.id}}}"
  }
]
```

Mustache templates

As you can see in the example, Mustache templates (`{{{name}}}`) can be placed anywhere within the inner `params` `Struct`. Through the templates you can fill fields of a REST request with any fields from previous responses. `0.result.id` for example is the field `result.id` of response belonging to the request with `Array` index `0`.

It is important to note, that a Mustache template entry cannot be replaced by an `Array` or `Struct`.

The method `Structs` are allowed to be a `String`. This is required, if you want to set other values than `String`s in the Mustache template (i. e. the mustache template cannot be place in double quotes). An example:

```
[
  "{\"method\": \"REST/GET my-rest-path\", \"params\": {}}",
  "{\"method\": \"REST/GET my-second-rest-path\", \"params\": {\"numerical-value\":
{{{0.result.propertyWithNumber}}} }}"
]
```