Serialization of Data between Client and API Interface

Both the Comet and the REST API interfaces share the same way of communication serialization using the JSON format. The only exception is calling SBA RPC methods which serve for sending and acquiring binary data from D2000 - in this case, the data is sent in a binary form. The JSON format is a universal format implemented almost in every programming language. Its advantage is not only native support in every browser but also easy readability for people since it is a text format. The disadvantage of the bigger capacity of a message in comparison with any binary format is minimalized by the use of a gzip compression implicitly supported by the REST and also the Comet API.

Serialization of the Unival Type

The basic unit of data change between a client and the D2000 system is the Unival type - clustering basic object attributes in D2000. The following example of the unival value in the JSON format represents a value of a real number type with the Valid state.

Example of writing a Unival value

```
{
  "type": "real",
  "value": 123.456,
  "status": ["Valid"]
}
```

Every Unival value has a clearly defined type with the use of the `type` attribute. In the case of sending Unival values into the D2000 system, the type attribute has to be always defined. The only exception is the possibility when we send instead of an object log of the Unival type the value directly, in our case 123,456. This value is automatically converted to the `"real"`, type by the SmartWeb server since it is a value with a decimal point (in the case of sending a string, the type would be automatically set to the `"text"` value and in the case of sending an integer to the `"int"` value.

Short notation of the Unival value

```
123.456
```

The answer from the SmartWeb server never comes in this shortened log but in the "object" log with a defined type (`type` attribute) and a value (value attribute) if it is valid.

Other attributes are not implicitly returned in the answer form the SmartWeb server because of optimization but a client can ask for them through a special attribute `returnAs` (below described).

The `type` attribute may gain the following values:

Value of the `type` attribute	Description of the value type
`"nan"`	none
`"bool"`	*VBool* type
`"int"`	integer
`"real"`	real number
`"station"`	*VStation* type
`"alarm"`	*VAlarm* type
`"process"`	*VProcess* type
`"time"`	integer (number of milliseconds from epoch)
`"timespan"`	real number (number of seconds)

`"text"`	text
`"array"`	field of objects *D2ApiValue*
`"qval"`	*VQval* type
`"record"`	two-dimensional field (row, column) of values

The list of all attributes of the Unival object is in the following table:

Attribute	Value type	Mandatory	Default value	Note
`type`	text	yes		
`value`	according to the type	no		unset attributes automatically mean invalid value, set attributes mean valid value (if the attribute is not overloaded in the status attribute)
`valueTime`	integer (number of milliseconds from epoch)	no	current time	time stamp
`valueTimes`	two-dimensional field (row, column) of integers (number of milliseconds from epoch)	no	current time	time stamps of values in a structure, only for the "record" type
`alarmTime`	integer (number of milliseconds from epoch)	no		time stamp of an alarm
`alarmTimes`	two-dimensional field (row, column) of integers (number of milliseconds from epoch)	no		time stamps of alarms in a structure, only for the "record" type
`flags`	field of texts (listed Flag type)	no	no flags	field of user flags, possible values are from `"A"` to `"P"`
`flagsSets`	two-dimensional field (row, column) of text fields (listed Flag type)	no	no flags	two-dimensional field of fields of user flags of structure values, only for the "record" type
`limitStatus`	text (listed LimitStatus type)	no	"InLimit"	limited state, possible values are: `"InLimit", "VeryLow", "Low", "High", "VeryHigh", "LimitsProblem"`
`limitStatuses`	two-dimensional field (row, column) of text fields (listed LimitStatus type)	no	"InLimit"	two-dimensional field of limited statuses of structure values, only for the "record" type
`processAlarmStatus`	text (listed ProcessAlarmStatus type)	no	"NoAlarm"	status of a process alarm, possible values are: `"NoAlarm", "ToOn", "ToOff", "On", "Off", "Err", "Oscillate", "ErrCmdOn", "ErrCmdOff", "SwToTrans", "SwToOff", "SwToOn", "SwToErr", "SwTrans", "SwOff", "SwOn", "SwErr", "ErrZalCmdOff", "HL", "VHL", "LL", "VLL", "ToHL", "ToVHL", "ToLL", "ToVLL", "ErrWriteCmd", "Change", "A29", "A30", "A31", "SysPrAl"`
`processAlarmStatuses`	two-dimensional field (row, column) of text fields (listed ProcessAlarmStatus type)	no	"NoAlarm"	two-dimensional field of statuses of process alarms ina structure, only for the "record" type
`status`	text fields (listed Status type)	no	"Valid"	status fields, possible values are: `"Valid", "ProcAlarm", "NoAckPAlarm", "PrAlSilent", "Weak", "NoAckValue", "Transient", "Default", "Manual", "AlCrit", "Unknown"`
`statusSets`	two-dimensional field (row, column) of text fields (listed Status type)	no	"Valid"	two-dimensional field of statuses of structure values, only for the "record" type
`formattedValue`	text	no		in the attribute, the formatted value of D2000 object returns in the answer from the server, unnecessary when calling RPC methods
`structType`	text	yes		structure name, only for the "record" type, mandatory on every "record" type sent to D2000
`definition`	*D2RecordDefinition* object	-		structure definition, only for the "record" type, set on every value with the "record" type returned from D2000
`returnAs`	text	no		have sense only when calling RPC methods with output parameters, defines logical name which will have the returned output value

`returnFields`	text fields	no	empty field	a special attribute defines additional returning attributes required by a client from the server
`returnTransformation`	*ReturnTransformation* object	no	null	only for the "record" type with one numerical column and growing time stamps of values, it contains the configuration of processing (downscaling) of a numerical column performed on a server before sending answers to a client, because of the big data range, values are described below

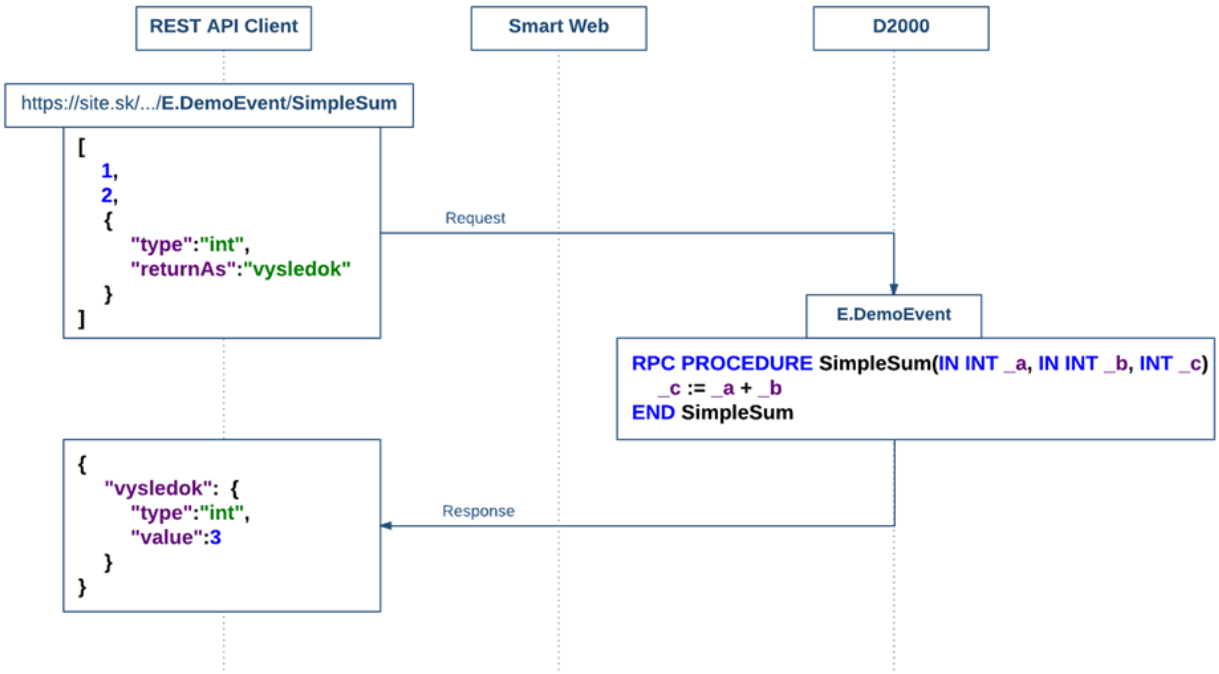
Implicit Conversion of Simple JSON Types to Unival Value

To simplify working through the API interface, it is possible to use simple types instead of Unival type objects as input values to the D2000 system. Such values have the validity attribute, the time of origin to the current time automatically set and other attributes gain their default value.

JSON format type	Corresponding Unival type	Note
`boolean`	`"bool"`	
`number`	`"bool"`	If the target type is in the D2000 *BOOL*. Numbers are interpreted according to the order in the *VBool* type.
`number`	`"int"`	If the target type is in the D2000 *INT*. If the given number was a real number, it will be rounded to an integer number.
`number`	`"real"`	If the target type is in the D2000 *REAL*.
`number`	`"time"`	If the target type is in the D2000 *TIME*. Warning: The number is interpreted as a number of seconds from 1972-01-01 00:00:00 UTC, not as a number of milliseconds from the epoch.
`string`	`"text"`	

Defining of Returned Values from RPC Methods

The attribute returnFields defines the logical name of an output parameter required by a client from the server.



In the figure, there is represented an example of the SimpleSum method calling by a client. The first two parameters are simple JSON types that are implicitly converted to right Unival values. The third parameter is an output one and the client defines in the `returnAs` attribute the logical name under which will be the output Unival value returned; in this case, it is the logical name `"vysledok"`.

Optimization of the Content of the Returned Unival Value

Because of the minimization of transmitted data, the Unival objects standardly contain on the output from D2000 only `type` and `value` attribute (if the value is valid). In the case of need of other extended value attributes, which are provided by the D2000 system, it is possible to set the `returnFields` attribute when calling and name required attributes in it. The value of the `returnFields` attribute has a format of a text field. Allowed values are in the following table.

Value in the `returnFields` field	Filled in attributes in response form server
`"AlarmTime"`	`alarmTime, alarmTimes`
`"Flags"`	`flags, flagsSets`
`"LimitStatus"`	`limitStatus, limitStatuses`
`"ProcessAlarmStatus"`	`processAlarmStatus, processAlarmStatuses`
`"Status"`	`status, statusSets`
`"ValueTime"`	`valueTime, valueTimes`
`"FormattedValue"`	`formattedValue`

Unival type Structure ("record")

The value of the structure type has its values and attributes stored as a two-dimensional field (row, column). Values gain the type according to the definition of structure in the D2000 system. It does not use extended attributes `alarmTime, flags, limitStatus, processAlarmStatus, status a valueTime`. Instead of them, it uses attributes `alarmTimes, flagsSets, limitStatuses, processAlarmStatuses, statusSets, valueTimes`, which are a two-dimensional field and components have the same type as the original attributes. Other attributes used only in structures are `definition` and `structType`. The `definition` attribute (*D2RecordDefinition* object) is always automatically set on all structural values that come from the D2000 system. This attribute describes columns' names and their types in a structured variable. Columns' types in a structure are a simplified version of Unival types.

Column type	Value type
`"bool"`	*VBool* type
`"integer"`	integer
`"real"`	real number
`"time"`	integer (number of milliseconds from epoch)
`"timespan"`	real number (number of seconds)
`"text"`	text
`"object"`	according to the type of referenced D2000 object

The `structType` attribute is a text attribute and defines the name of the D2000 object of the structure definition type. It is mandatory to set it for every structured value that is inputted into the D2000 system.

Príklad posielanéj hodnoty typu štruktúra - definícia SD.ArrReal_Text, 2 stpce (int, text), 3 riadky:

```
{
  "type": "record",
  "structType": "SD.Arr_Real_Text",
  "value": [[1, "One"], [2, "Two"], [3, "Three"]]
}
```

Príklad vracanej hodnoty typu štruktúra - definícia SD.ArrReal_Text, 2 stpce (int, text), 3 riadky:

```
{
  "type": "record",
  "definition": {
    "columnTypes": ["integer", "text"],
    "columnNames": ["digit", "name"]
  },
  "value": [[1, "One"], [2, "Two"], [3, "Three"]]
}
```

Transformations of Time Rows

Time row is a one-column structure in which values are real numbers and have increasing time stamps set. When displaying time rows, all values from requested interval are often not necessary or required (especially because of performance). For such needs, transformations of time rows are implemented which decrease the number of transmitted data. It is possible to request the transformation by setting the `transformation` attribute on the output structured value.

Downsampling

Downsampling is reducing the number of values according to the given step or to the given number of values. The used algorithm tries as much as possible to keep the course of the curve of the original time row and does not smooth the final curve as the common resampling using averaging.

Vyžiadané zredukovanie algoritmom Largest Triangle Three Buckets na 100 hodnôt

```
{
  "type": "record",
  "structType": "SD.Arr_Real",
  "returnTransformation": {
    "type": "lttb",
    "threshold": 100
  }
}
```

Vyžiadané zredukovanie algoritmom Largest Triangle Three Buckets s krokom 86400 sekúnd (1 de)

```
{
  "type": "record",
  "structType": "SD.Arr_Real",
  "returnTransformation": {
    "type": "lttb",
    "step": 86400
  }
}
```

OHLC

The OHLC (Open-High-Low-Close) algorithm transforms the output time row in a way that for every interval it finds first, maximal, minimal and last value. Optionally when using this algorithm, it is possible to set whether the intervals should be continuous (successive intervals have the first and the last value common - by default) or discrete - the `discrete` attribute and where should be the time stamp of the value representing the whole interval placed (the beginning "Start" or in the middle "Midpoint" - by default) - the `timestampPlacement` attribute.

Vyžiadané OHLC transformácie s krokom 86400 sekúnd (1 de) s diskrétnymi intervalmi a asovými znakami na zaiatku intervalu

```
{
  "type": "record",
  "structType": "SD.Arr_Real",
  "returnTransformation": {
    "type": "ohlc",
    "step": 86400,
    "discrete": true,
    "timestampPlacement": "Start"
  }
}
```