Journal of Theoretical and Applied Information Technology
15th October 2016. Vol.92. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

INTEGRATION OF MEDICAL SIMULATION EQUIPMENT INTO UNIFIED DATA SYSTEM

[1]ANDREY ALEKSEEVICH SVISTUNOV, [1]DENIS MIHAYLOVICH GRIBKOV, [2]SERGEY SERGEEVICH SMIRNOV, [2]DMITRII ALEKSANDROVICH SYTNIK, [3]ALEKSANDR LVOVICH KOLYSH

[1]State Federal-Funded Educational Institution of Higher Professional Training I.M. Sechenov First Moscow State Medical University of the Ministry of Health of the Russian Federation, 119991, Moscow, Trubetskaya Street, 8, block 2, Russia
[2]Complex Systems LLC, 170021, Tver, Skvortsova-Stepanova Street, 83, Russia
[3]INTERMEDICA LLC, 603005, Nizhny Novgorod, Semashko Street, 20, Russia

ABSTRACT

At present there are no unified standards, regulations, protocols of data transfer from medical simulators by various manufacturers. This prevents development and improvement of automation systems of simulation training. This work is aimed at development of software which will provide merging of medical simulators and training devices by various manufacturers into unified data system. This target is based on developed and documented binary data format, where data are presented in the form of coupled key/value pair. On the basis of the developed binary data format the structure of data package has been developed as well as command list for interaction between client and server applications. Using the developed data format and TCP/IP proprietary data exchange protocol has been developed. As an alternative, the data exchange protocol on the basis of JSON format has been developed. The developed client and server applications facilitate data exchange between software of medical simulation equipment and designed data system. The developed software has been tested on the following medical simulators and training devices: Resusci Anne (Laerdal), Lap-X (Epona), and Lap Mentor (Simbionix). In the future it is planned to expand possibilities of the data system and to connect new simulators and training devices to this system. The protocol and software, developed in this work, make it possible to combine data from various medical simulators in the frame of unified data system in real time. The obtained results facilitate automation of training processes on the basis of medical simulators.

Keywords: *Simulator, Simulation Training, Medical Training Device, Data Base, Software, Data System*

1. INTRODUCTION

Simulation training in medicine becomes more and more popular nowadays [1]. Simulation centers are being established aiming at education of students, post graduates, medical residents, as well as further training of practicing physicians [2]. Medical simulation equipment and medical training devices in such centers are supplied by various manufacturers for various fields: anesthesiology and emergency medicine [3], obstetrics, gynecology, perinatology and pediatry [4], surgery [5]. This list increases constantly in the course of development and improvement of simulation techniques. Currently there are several data systems available in the market and aimed at automation and management of training in simulation centers. However, the market of this field of data systems is not sufficient, there are no unified requirements and standards, regulations, protocols of data transfer from medical simulators by various manufacturers. Thus, manufacturers of simulators develop software only for their own products and prevent communications with simulators by other manufacturers. This results in significant obstacles for operation of simulation centers, when technical achievements cannot improve efficiency, expected initially, but rather lead to creation of additional processes and tasks. This can be exemplified by such manufactures and their software: CAE Healthcare LearningSpace [6], Laerdal SimCenter [7], Simbionix 3D Systems [8].

1.1.Formulation of the problem

Each manufacturer of medical simulators has its own approach to designing of interface, arrangement of data exchange and data storage means. For instance, data about users and test results can be stored in various data base management systems (DBMS), such as MSAccess, MSSQLServer, SQLite, MySQL. In addition, data can be stored in XML format [9] and arbitrary binary formats. Software of certain simulators and training devices arranges the output data on results

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of system operation in the form of reports in PDF format. The test results are not always presented in Russian language and are not always arranged according to Russian medical standards. Therefore, a complicated problem appears concerning integration of required data formats of simulation equipment by various manufacturers into unified data system.

The training devices summarized in Table 1 were selected for integration into unified data system. The selection was based on the principle of diversity with regard to medical fields and manufacturers.

Table. 1. Medical Training Devices And Simulators Selected For Integration Into Unified Data System

Designation	Manufacturing company	Link to website
Resusci Anne	Laerdal	[10]
Lap-X	Epona	[11]
Lap Mentor	Simbionix	[12]
Gi Mentor	Simbionix	[13]
ORZone	ORZone	[14]
Angio Mentor	Simbionix	[15]
Arthrosim	Touch of Life Technologies	[16]
UroSim, ArthroS, HystSim	VirtaMed	[17] [18] [19]

In order to integrate medical simulation equipment into unified data system certain data exchange protocol is required, which should be developed. Hence, a format of transfer of data and arbitrary objects should be selected. In our case the objects can be presented by tables with various data, video files, binary files with plots, various images. In this regard such widely applied formats as XML [19] and JSON [20] are not suitable for our case. Network communication using these formats is very resource intensive and in some cases it cannot be applied at all. Application of binary specifications XML and JSON is not very convenient due to different implementation and, as a consequence, to different data interpretation. In addition, the aspect of implementing complexity arises, since one formats enable increase in productivity and the other provides possibility of more compact data presentation due to complex specification. In this regard we propose binary format of data exchange (hereinafter referred as the Format) as the most suitable for this situation.

2. EXPERIMENTAL SETUP

2.1 Data format specification

For any data exchange format the data types should be defined which will be supported by such format. For data transfer stored in data bases it is proposed to use format in the form of identifier/data. The format can be presented as follows:

(Data type)(Data identifier)(Data length)(Data)

Let us consider this Format in details. “Data type” is one byte with the value from -127 to 128. This mandatory parameter is unique identifier of data type. The data type value can be negative. In this case the “Data” field contains array. The data type is followed by “Data identifier” – a string in ASCII format [21] terminated with null byte. The “Data identifier” should be unique and not duplicated in one data package, since this can lead to ambiguous definition of this or that value. The “Data identifier” is followed by “Data length” in bytes. For binary objects the “Data length” can be in the range from 0 to 2147483647 and is a four-byte unsigned integer value. This field is not a mandatory parameter and can be absent, whereas the value of “Data length” field is defined according to data type. The data can assume any values according to identifier of data type. Therefore, description of designation of data base field and its value are described by the pair: field/value. Taking into account that there are simple data types, the length of which is fixed in bytes, the “Data length” field can be absent which reduces the networking packet size. In addition, the time for data unpacking is also reduced. In the Format it is proposed to use one-byte, two-byte, four-byte integer numbers, signed and unsigned, single- and double-precision floating-point numbers, logical type, as well as strings and arbitrary binary objects. Little-endian byte order is assumed in this format [22].

2.2 Simple types

In the Format it is proposed to use simple data types, which are supported in numerous programming languages. Table 2 summarizes identifiers of data types, their names in library, size in bytes and description.

Journal of Theoretical and Applied Information Technology
15th October 2016. Vol.92. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table.2. Identifiers of data types

Data type identifier	Name in library	Data length in bytes	Data type description
0	dt_null	0	Null identifier
1	dt_bool	1	Logical (Boolean) data type with two possible values: (true) and (false)
2	dt_sint8	1	Signed integer data type
3	dt_uint8	1	Unsigned integer data type
4	dt_sint16	2	Signed integer data type
5	dt_uint16	2	Unsigned integer data type
6	dt_sint32	4	Signed integer data type
7	dt_uint32	4	Unsigned integer data type
8	dt_sint64	8	Signed integer data type
9	dt_uint64	8	Unsigned integer data type
10	dt_float	4	Single-precision floating-point number
11	dt_double	8	Double-precision floating-point number
12	dt_string	up to 2147483647	String
13	dt_object	up to 2147483647	Binary object

Logical or Boolean type is the simple data type which can assume two possible values: true or false. This data type presents in nearly all programming languages, it is also used in data bases. One bit is sufficient for storage of this type, however, minimum referenced storage in one byte is usually used. In certain programming languages it is assumed that 1 is true and 0 is false. In the proposed Format we also will adhere to this provision. Integer data types are simple data, they are widely applied in programming languages. They serve for presentation of integer numbers. The set of numbers of these types is comprised of subsets of integer numbers. Each integer type is limited by minimum and maximum values. The integer data types are subdivided into signed and unsigned ones. Table 3 summarizes identifiers of unsigned integer data types and their range.

Table.3. Identifiers Of Unsigned Integer Types And Their Ranges

Name in library	Length in bytes (bits)	Data range
dt_uint8	1(8)	$0..2^8-1$
dt_uint16	2(16)	$0..2^{16}-1$
dt_uint32	4(32)	$0..2^{32}-1$
dt_uint64	8(64)	$0..2^{64}-1$

Signed integer types are also used together with unsigned integer types. Sign is usually defined by most significant bit: if the most significant bit is unit, then the number is considered as negative. Table 4 summarizes identifiers of signed integer data types and their ranges.

Table.4. Identifiers Of Signed Integer Data Types And Their Ranges

Name in library object (identifier)	Length in bytes (bits)	Data range
dt_sint8	1(8)	$-2^7..2^7-1$
dt_sint16	2(16)	$-2^{15}..2^{15}-1$
dt_sint32	4(32)	$-2^{31}..2^{31}-1$
dt_sint64	8(64)	$-2^{63}..2^{63}-1$

The floating-point numbers are real numbers. A number is stored in the form of mantissa and exponent, herewith, floating-point number has fixed relative and varying absolute accuracy. In the Format it is proposed to apply four-byte and eight-byte floating point numbers.

2.3 Example of presentation of simple data type

Data types, corresponding to data type identifiers from 1 to 10 (Table 2), are simple. For their use the "Data length" field is not required. "Data length" in this case is defined according to Table 5.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table. 5. Length Of Primitive Data Types

Data type	Length in bytes
dt_null	1
dt_bool	1
dt_sint8	1
dt_uint8	1
dt_sint16	2
dt_uint16	2
dt_sint32	4
dt_uint32	4
dt_sint64	8
dt_uint64	8
dt_float	4
dt_double	8

The data record format corresponding to identifiers of data types from 1 to 11 (Table 2) is proposed to be as follows:

(Data type)(Data identifier)(Data)

Data type is defined according to Table 2 and is one byte. The data identifier is a string in ASCII format [21], its length can be in the range from 1 to 2147483647 bytes. This string should be terminated with null byte. Thus, in order to locate simple data types in this format we would use additionally only two bytes: data type identifier and null byte at the end of data identifier. For instance, let us locate the data from Table 5 into this format. Data type will be considered as data identifier and the length in bytes as data. We obtain the following:

[0][dt_null0][b]

[1][dt_bool0][b]

[2][dt_sint80][b]

[3][dt_uint80][b]

[4][dt_sint160][bb]

[5][dt_uint160][bb]

[6][dt_sint320][bbbb]

[7][dt_uint320][bbbb]

[8][dt_sint640][bbbbbbbb]

[9][dt_uint640][bbbbbbbb]

[10][dt_float0][bbbb]

[11][dt_double0][bbbbbbbb]

In this example the data identifier is terminated with null byte. Instead of the data value the number of bytes occupied by the type is mentioned.

2.4 Strings

Strings are presented by ASCII format [21], the string end should mandatory contain null byte. This byte defines the string end. Table 2 summarizes string identifier, its name in library and maximum number of symbols. Format for serialization [23] of string data will be as follows:

(dt_string)(Identifier0)(Data0).

The string should be terminated with null byte. As an example let us locate string variable Name into this format, its value is “FirstName”:

(dt_string)(Name0)(FirstName0)

This example illustrates that the string data can be wrapped by only three bytes irrespective of its length.

2.5 Binary objects

Binary objects are presented by the following format:

(Data types)(Data identifier)(Data length)(Data).

The “Data length” field is mandatory in presentation of binary objects. Maximum length of binary object, similar to strings, is 2147483647 bytes. Table 2 shows data type identifier for binary object. As an example of binary object the previous case can be considered:

(dt_object)(Name0)(bbbb)(FirstName)

The difference from previous example is that for binary objects, irrespective of their length, the “Data length” field is defined, which occupies four bytes. In this example each byte is denoted as “b”. Therefore, binary data can be wrapped by six bytes. It should be noted that the “FirstName” string in this case is considered as sequence (array) of bytes, the length of which is mentioned in the “Data length” field, hence, there is no null byte in the end. It is possible not to include the notion of binary object in the Format description but to apply only arrays. However, serialization [23] may be required for array containing data arrays as elements. In such

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

situation we can easily apply array of binary objects.

2.6 Arrays

Each data type corresponds to data array. In order to indicate that "Data" field contains array of certain data type it is necessary to set negative identifier for data type. Table 6 summarizes identifiers for arrays of various data types.

Table. 6. Identifiers Of Arrays

Identifier of array type	Data type in array
-1	dt_bool
-2	dt_sint8
-3	dt_rint8
-4	dt_sint16
-5	dt_uint16
-6	dt_sint32
-7	dt_uint32
-8	dt_sint64
-9	dt_uint64
-10	dt_float
-11	dt_double
-12	dt_string
-13	dt_object

As can be seen in the table, absolute values of identifiers of array types correspond to data type in the array. Format for wrapper of any array is as follows:

(Data type)(Data identifier)(Data length)(Data)

The "Data type" field will identify value type in the "Data" field. This is provided by existence of unity in the most significant bit of data type identifier. Therefore, in this format for each data type arrays of these types are defined. While using string array it is necessary to add null byte in the string end in order to separate strings. When array of binary objects is used each block should be preceded by indication of its length, which consists of four bytes. This is an example of array of binary objects:

(-Data type)(Type identifier)(Data length)(<Data length>{object}<Data length>{object}…)

2.7 Description of data package

Data exchange between client and server applications is performed by means of data packages, their format is as follows:

(PackageLen)

(dt_string)(Command)(valueCommand)

(dt_string)(SimulatorName)(valueSimulatorName)

(dt_object)(Data)(LenData)(valueData)

The "PackageLen" field defines package length in bytes. The size of "PackageLen" field is four bytes. The "Command" field contains command, the "SimulatorName" field contains simulator unique name. The "Data" field contains arbitrary data. The "Data" field is not mandatory and can be absent, depending on command. The field sequence "Command", "SimulatorName" and "Data" has no name. Description of the fields of data package is given in Table 7.

Table.7. Description Of Data Package

Field	Assignment	Comments
PackageLen	Package length in dt_uint32 format	Mandatory parameter
Command	Command	Mandatory parameter
Simulator Name	Unique simulator designation	Mandatory parameter
Data	Transferred data	May be absent

The "Command" field contains command which should be supported by client and server application. Table 8 summarizes the commands to be supported by the applications. The "GetNewUser" command is intended mainly for the server. The client sends this command to the server in order to obtain data about new user. If new users exist, then the server transfer data to the client concerning this user by means of "SetNewUser" command. If there are no new users, then the server transfer "NoNewUser" command to the client, in this case the client stops request of new users. The "SetResult" command is sent by the client to the server in order to store the results on the server. If data were successfully stored on the server, then the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

server sends the "OkSetResult" command to the client. If an error occurred while saving data on the server, or data were not saved by any reason, then the server sends the "NoSetResult" command to the client. In this case the client attempts to send the "SetResult" command to the server once more. Table 8 summarizes the command list.

Table.8. Command List

Command	Assignment
GetNew User	Query of new users of the system
SetNew User	Response for GetNewUser command with new users
NoNew User	Response for GetNewUser command without new users
SetResult	Entry of exam results into data base
OkSetResult	Response for SetResult command after successful data entry into data base
NoSetResult	Response for SetResult command after unsuccessful data entry into data base

2.8 Examples of data packages

A data package for results transfer has the following form:

(bbbb)

(12)(Command0)(SetResult0)

(12)(SimulatorName0)(Simulator10)

(13)(Data0)(LenData)(Value)

The first four bytes are the package length. Then, the command to transfer data into common base, simulator name. The "Data" field contains data about exam results. Existence of null byte define the string end. Data package for transfer of user records has the following form:

(bbbb)

(12)(Command0)(SetUser0)

(12)(SimulatorName0)(Simulator10)

(13)(Data0)(LenData)(Value)

The first four bytes are the package length. Then, the command to record user into the data base of simulator, simulator name. The "Data" field contains data about user records.

2.9 Description of data package on the basis of JSON format

The main distinction of the JSON format [20] from binary formats is that data are transferred in text format. This format is easy-to-read, but has poorer productivity in comparison with binary formats. Since this format is widely applied, it is also supported by our system. Data transfer protocol using the JSON format is as follows:

{"Command":"Value",

"SimulatorName":"Value",

"Data":{"StringField1":"Value1",

"IntegerField1":123,

"FloatField1":12.3,

"Array":[1,2,3,4,5]}}

The "Command" field contains command name (Table 8). The "SimulatorName" field contains unique name of training device. The "Data" field contains data about fields and their values. In the example "StringField1" is string field, "IntegerField1" is integer type field, "FloatField1" is floating point number field, "Array" is value array. For instance, , it is required to write "Score" field into the JSON format, its value is described by single-precision floating point number:

{"Score":123.4}

In this example we used five symbols to wrap the data. Now let us write the "FirstName" field into the JSON format, its value is "Ivan". This is written as follows:

{"FirstName":"Ivan"}

This example illustrates seven symbols are required for string data.

Let us consider addition of integer data of 100 pieces by 4 bytes. For instance, time array upon artificial lung ventilation. The array size in binary form is 400 bytes. If the wrapper for binary data of the proposed binary formats is added, then it is 406 bytes. If the name length is included, then we have 409 bytes in binary format. Now let us analyze the possible results in the JSON format. Times are recorded into file in milliseconds, and the exam time may be several minutes. The value array will be as follows: [1000000,1000020,…], therefore,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

one value requires about seven symbols in average, and if points are considered, then about eight symbols. Finally we have 800 bytes. This is by two times more than in binary presentation.

3. RESULTS

As a result of the performed work we developed and documented binary format for serialization of data and arbitrary objects. On the basis of this Format and TCP/IP we developed proprietary data exchange protocol. As a an alternative, protocol on the basis of JSON format was developed [20]. Unfortunately, it is characterized by more modest possibilities than binary format. Client/server application was developed on the basis of protocol based on binary format, which facilitates recording of users in simulator software, as well as collecting necessary data with their storage in common data base. It is proposed to integrate medical simulators, summarized in Table 1, into this system. At this stage we tested software complex for the following simulators: Resusci Anne [10], Lap-X [11], and Lap Mentor [12]. Selection of these simulators and training devices was stipulated by the fact that they use various data base management systems. Client application provides possibility to collect data about results and to store them in local data base as packages. If connection to the server is active, then the client sends the results to the server and requests data about new users. The server receives data and stores them in common data base of the system. A result of operation of this system is data transfer from simulator software into common data base, as well as user registration in local software of simulator.

4. DISCUSSION

At present there are numerous binary formats on the basis of which is it possible to arrange package for data exchange, though, each of them has intrinsic disadvantages in terms of implementation of our task. Table 9 summarizes binary data format and references to their description. Some of them support not all data types necessary for our case. Some formats have no special form for data recording as key/value.

Table.9. References To Binary Formats

Designation of data format	Reference
BSON	[24][25][26]
BJSON	[27]
UBJSON	[28][29][30]
SmileFormat	[31][32]
Protocol Buffers	[33]

For illustrative purposes it is possible to compare our developed format of data presentation with the JSON format and the most suitable for our purposes UBJSON format [29]. General view of byte structure UBJSON is as follows:

(type)(length)(data)

"Type" is 1 byte, ASCII symbol [21]. It is used for indication of data type following it. "Length" (option) is 1 or 4 bytes (integer) depending on the object length or size. For array: its length, for object: the number of key/value pairs. If the length or number of elements is from 0 to 254 inclusively, then 1 byte is used. This field with the value of 255 is reserved for objects and arrays of unknown length. "Data" (option) is the byte sequence, directly representing the object data. The "length" and "data" fields are used or not used depending on the data type. For instance, 32-bit integer type has standard size of 4 bytes. In order to record this type 1 byte is required for indication of type and 4 byte for the value. In this case the "length" field is not used since it is not required. The package structure, for instance, for UBJSON format will be as follows:

(bbbb)

(type)(b)(Command)(type)(b)(Value)

(type)(b)(SimulatorName)(type)(b)(Simulator1)

(type)(b)(Data)(type)(bbbb)(Value)

It can be seen in this example that each string requires for two bytes, and if a string is longer than 255, then five bytes. Respectively, upon network transfer of field and its value the overhead will be seven bytes. Table 10 shows the values of network load for each of three formats. Comparison was performed for all data types in the Format.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table.10. Comparison Of Network Load For Three Data Formats

Data type	JSON (length in bytes)	BJSON (length in bytes)	Proposed format (length in bytes)
dt_bool	8-9	5	4
dt_sint8	5-7	5	4
dt_uint8	5-7	5	4
dt_sint16	5-10	6	5
dt_uint16	5-9	6	5
dt_sint32	5-15	8	7
dt_uint32	5-14	8	7
dt_sint64	5-20	12	11
dt_uint64	5-19	12	11
dt_float	5-12	8	7
dt_double	5-20	8	7
dt_string (length<255)	from 6	from 5	from 4
dt_string (length>255)	from 6	from 8	from 4
dt_blob (length<255)	from 6	from 5	from 4
dt_blob (length>255)	from 6	from 8	from 4

5. CONCLUSIONS

The protocol and software, developed in this work, make it possible to combine data from various medical simulators in the frame of unified data system in real time. The obtained results facilitate automation of training processes on the basis of medical simulators.

The developed data format in terms of network load does not vary noticeably from its binary analogs. It follows from Table 10 that the highest network load will be applied by protocol on the basis of JSON format. After connection of a simulator to the unified system it is possible to use any languages, Russian in our case, thus improving data perception. On the basis of problem analysis during data system development we propose data exchange protocol which will allow data transfer of various types in unified structure.

The designed data system has wide possibilities for its application in simulation training centers in Russia. The advantages of this system consist of its flexibility and versatility, and the developed exchange protocol facilitates significant increase in data transfer and processing. In the future the list of medical simulators and training devices summarized in Table 1 is supposed to be expanded. We plan to increase possibilities of the data system and to connect new simulators and training devices to this system.

6. ACKNOWLEDGMENTS

The work was supported by Ministry of education and science of Russian Federation (Grant 14.607.21.0130, unique identifier of applied researches: RFMEFI60715X0130).

REFERENCES:

[1] Simulation Training in Medicine, ROSOMED Russian Society for Simulation Training in Medicine. Ed. by Svistunov A. A. Compiled by Gorshkov M. D. Publishing House of the First Moscow State Medical University; Moscow 2013

[2] Balkizov Z. Z. (2011). Continuous medical training. Application of simulation technologies in health care center. Zdravookhranenie, 10 (44-49)

[3] Simulation Training in Anesthesiology and Emergency Medicine / compiled by M. D. Gorshkov; ed. by V. V. Moroz and E. A. Evdokimov.-Moscow: GEOTAR-Media, 2014.

[4] Simulation Training in Obstetrics, Gynecology, Perinatology and Pediatry / compiled by M. D. Gorshkov; ed. by G. T. Sukhikh.- Moscow: ROSOMED, 2015.

[5] Simulation Training in Surgery / ed. by V. A. Kubyshkin, S. I. Emel'yanov, and M. D. Gorshkov.- Moscow: GEOTAR-Media, 2014.

[6] CAE Healthcare LearningSpace. Date Views 16.06.2016 www.caehealthcare.com/eng/audiovisual-solutions/learning-space.

[7] Laerdal SimCenter. Date Views 16.06.2016 www.laerdal.com/SimCenter.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[8] Simbionix 3D Systems. Date Views 16.06.2016 www.simbionix.com/mentorlearn/.
[9] VirtaMed UroSim. Date Wiews 16.06.2016 www.virtamed.com/en/medical-training-simulators/urosim/.
[10] Resusci Anne. Date Wiews 16.06.2016 www.laerdal.com/ResusciAnne/.
[11] Lap-X. Date Wiews 16.06.2016 www.medical-x.com/products/lap_x/.
[12] Lap MENTOR. Date Wiews 16.06.2016 www.simbionix.com/simulators/lap-mentor/.
[13] GI MENTOR. Date Wiews 16.06.2016 www.simbionix.com/simulators/gi-mentor/.
[14] ORCAMP-Simulation. Date Wiews 16.06.2016 www.orzone.com/#!simulation/c1uks/.
[15] ANGIO MENTOR. Date Wiews 16.06.2016 www.simbionix.com/simulators/angio-mentor/.
[16] ArthroSim Arthroscopy Simulator. Date Wiews 16.06.2016 www.toltech.net/medical-simulators/products/arthrosim-arthroscopy-simulator/.
[17] VirtaMed ArthroS. Date Wiews 16.06.2016 www.virtamed.com/en/medical-training-simulators/arthros/.
[18] VirtaMed HystSim. Date Wiews 16.06.2016 www.virtamed.com/en/medical-training-simulators/hystsim/
[19] XML. Date Wiews 16.06.2016 www.ru.wikipedia.org/wiki/XML/.
[20] JSON. Date Wiews 16.06.2016 www.ru.wikipedia.org/wiki/JSON/.
[21] ASCII. Date Wiews 16.06.2016 www.ru.wikipedia.org/wiki/ASCII/.
[22] Byte Order. Date Views 16.06.2016 www.ru.wikipedia.org/wiki/Порядок_байтов/.
[23] Serialization. Date Wiews 16.06.2016 www.ru.wikipedia.org/wiki/Сериализация/.
[24] BSON. Date Wiews 16.06.2016 www.ru.wikipedia.org/wiki/BSON/.
[25] BSON. Date Wiews 16.06.2016 www.bsonspec.org/.
[26] BSON Types. Date Wiews 16.06.2016 www.docs.mongodb.com/manual/reference/bson-types/.
[27] BJSON. Date Wiews 16.06.2016 www.bjson.org/.
[28] UBJSON. Date Wiews 16.06.2016 www.en.wikipedia.org/wiki/UBJSON/.
[29] Universal Binary JSON Specification. Date Wiews 16.06.2016 www.ubjson.org/.
[30] Universale Binary JSON 0.9-dev documentation. Date Wiews 16.06.2016 www.universal-binary-json.readthedocs.io/en/latest/.
[31] SmileFormat. Date Wiews 16.06.2016 www.wiki.fasterxml.com/SmileFormat/.
[32] SmileFormatSpec. Date Wiews 16.06.2016 www.wiki.fasterxml.com/SmileFormatSpec/.
[33] Protocol Buffers ru. Date Wiews 16.06.2016 www.wikipedia.org/wiki/Protocol_Buffers/.