VoltDB Client Wire Protocol

VoltDB Team - 6/27/2011 - VoltDB Client Wire Protocol Version 0

Overview

A client connection to a VoltDB instance consists of a TCP connection on port 21212. After the initial login process the only exchange between the client library and the VoltDB server is the invocation of and response to stored procedures. All messages in the VoltDB wire protocol are big endian and all integers are signed. Every message is length preceded with a 4 byte integer and the length does not included the length value. The first value after the length preceding value is the version number of the wire protocol represented as a byte. These two items precede all messages.

The login message is the first message the client library sends to the VoltDB server and it is required even if authentication is disabled in the server's configuration. The login message consists of a service name string, a username string and a 160-bit SHA-1 hash of the password. The response from the server consists of a response byte. A value of 0 indicates successful authentication and all other values indicate failure. If authentication fails the server will close the connection. The client can safely send invocations before receiving an authentication response.

The procedure invocation request contains the procedure to be called by name, and the serialized parameters to the procedure. The message also includes an opaque 8 byte piece of client data that will be returned with the response, and can be used by the client to correlate requests with responses.

The returned response contains a byte status code, an integer measuring intra-cluster latency, a serialized exception if an error occured and the exception was serializable, an array of VoltTables (may be length 0, never null), a string value containing any extra information the server included, and the 8 byte piece of client data contained in the originating procedure invocation.

The following sections describe how values are serialized. The wire protocol is still under development and will be adapted to support new authentication methods and new data types as necessary.

Basic Data Types

Binary fields do not have a wire type associated with them but they are present in certain messages. Binary fields (opacque client data, hashes) do not have any endianness, sign, or size other then what is specified in the message format.

Integer Types:

All integer types are signed, twos-compliment and big-endian.

- Byte - 1 Byte
- Short - 2 Bytes
- Integer - 4 Bytes
- Long - 8 Bytes

Floating Point Type

Only 8-Byte Double types are supported using the byte representation in IEEE 754 "double format." Positive and negative infinity, as well as NaN, are supported on the wire, but not guaranteed to work as SQL values.

String Type

Strings begin with a 4-byte integer storing the number of bytes of character data, followed by the character data. UTF-8 is the only supported character encoding. Note: Strings are artificially limited to 1 megabyte. The NULL string has a length preceded value of -1 (negative one) followed by 0 (zero) bytes of string data. The empty string is represented with a length preceding value of 0.

A String encoded into a parameter set has to be deserialized as a Java String before being passed to a stored procedure and this is a relatively slow operation. If the stored procedure does not need the Java String representation it can specify a byte array as its argument instead of String. The String should then be presented in the parameter set as a byte array (not preceded with the String wire type).

If an array of bytes is passed as a parameter to a SQL statement in a stored procedure, and that parameter expects a string, it will automatically be converted to a string on the native side without any Java deserialization. SQL statements do not support array parameters so this is not ambigous. In the current implementation this can be a signficant performance win, especially with large parameter sets with many strings.

The following table presents the byte by byte serialization of the string "foo"

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
00	0	String Length	3	String is 3 bytes long
01	0			
02	0			
03	3			
04	102	Characters	f	String data is "foo"
05	111		o	
06	111		o	

Varbinary Type

Varbinary is a effectively a binary string, using the same serialization and storage.

Like strings, varbinary begin with a 4-byte integer storing the number of bytes of raw data, followed by the raw data itself. Note: Like strings, varbinary values are artificially limited to 1 megabyte. The NULL varbinary has a length preceded value of -1 (negative one) followed by 0 (zero) bytes of data. The value of zero bytes is represented with a length preceding value of 0.

The native java type for varbinary is byte[]. Stored procedures and SQL statements will accept byte[] as input for varbinary parameters. For compatibility with textual SQL and less binary-friendly clients, varbinary parameters may also be passed as hexidecimal strings using standard string serialization and the string type code indicator. For example, the string "aa" would represent a single byte of value 170. The hex-encoding is case-insensitive and will fail on invalid input, such as odd-length strings.

Date Type

All dates are represented on the wire as Long values. This signed number represents the number of microseconds (not the usual milliseconds before or after Jan. 1 1970 00:00:00 GMT, the Unix epoch.

Decimal Type

VoltDB implements a fixed precision and scale DECIMAL(38,12) type. This type is serialized as a 128 bit signed twos complement integer reperesenting the unscaled decimal value. The integer must be in big-endian

byte order with the most significant bytes first. Null is serialized as the smallest representable value which is "-170141183460469231731687303715884105728." Serializing values (Null excluded) that are greater than "99999999999999999999999999999999999999" or less than "-99999999999999999999999999999999999999" will result in an error response.

Note that the serialization is given above in terms of a 128 bit integer. Since the logical value represented is actually scalled by 12 decimal places, the logical maximum value, minimum value and null values are "99999999999999999999999999.999999999999", "99999999999999999999999999.999999999999" and "170141183460469231731687303.715884105728" respectively.

The following table presents the byte by byte serialization of the number "-23325.23425"

Byte Offset	Byte Value (dec)
00	-1
01	-1
02	-1
03	-1
04	-1
05	-1
06	-1
07	-1
08	-1
09	-83
10	33
11	-46
12	-78
13	57
14	-39
15	-128

Application Specific Data Types

A Byte value specifying the type of a serialized value on the wire.

- ARRAY = -99
- NULL = 1
- TINYINT = 3
- SMALLINT = 4
- INTEGER = 5
- BIGINT = 6
- FLOAT = 8
- STRING = 9
- TIMESTAMP = 11

- DECIMAL = 22
- VARBINARY = 25

Procedure Call Status Code

A Byte value specifying the success or failure of a remote stored procedure call.

- SUCCESS = 1
- USER_ABORT = -1
- GRACEFUL_FAILURE = -2
- UNEXPECTED_FAILURE = -3
- CONNECTION_LOST = -4

Compound Data Types

Array Types

Arrays are represented as Byte value indicating the wire type of the elements and a 2 byte Short value indicating the number of elements in the array, followed by the specified number of elements. The length preceding value for the TINYINT (byte) type is length preceded by a 4 byte integer instead of a 2 byte short. This important exception allows large quantities of binary or string data to be passed as a byte array. The size of byte arrays is artificially limited to 1 megabyte. Each array is serialized according to its type (Strings as Strings, VoltTables as VoltTables, Integers as Integers). Arrays are only present as parameters in parameter sets.

- Size is limited to 32,767 values due to the signed short length with the exception of TINYINT (byte) arrays which use a 4 byte integer length and are limited to 1 megabyte.
- All values must be homogeneous with respect to type.

The following example serialization shows an array with two String elements ("foo1", "foo2").

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
00	9	Element Type	9	Array Elements are Strings
01	0	Element Count	2	Array contains two elements
02	2			
03	0	String Length	4	String is 4 bytes long
04	0			
05	0			
06	4			
07	102	Characters	f	String data is "foo1"
08	111		o	
09	111		o	
10	49		1	
11	0	String Length	4	String is 4 bytes long

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
12	0			
13	0			
14	4			
15	102	Characters	f	String data is "foo2"
16	111		o	
17	111		o	
18	50		2	

Complex API Data Types

VoltTable

On the wire a VoltTable is serialized as a header followed by tuple data. VoltTables, like all VoltDB serialized structures are stored in network byte order.

(Note: In the following description, the term "array" means a sequence of objects of the specified type, not a VoltDB array object as described in the previous section. Because the number of columns is fixed and is already part of the VoltTable descriptor, we know how many objects are in each array and a full array descriptor is not needed.)

It should also be noted that although a description of the VoltTable structure is being provided here for completeness, in most cases the client interface does not need to interpret the structure, but rather passes it unchanged between the server and the client application.

Name	Type	Length (bytes)	[Basic | Compound | Complex]			
Total table length	Integer	4	Basic			
Table Metadata Length	Integer	4	Basic			
Status Code	Byte	1	Basic			
Column Count	Short	2	Basic			
Column Types	Array of Bytes	variable	Compound			
Column Names	Array of Strings	variable	Compound			
Row Count	Integer	4	Basic			
Row Data...						

Notes on the Header Format:

- The "Table Metadata Length" stores the length in bytes of the contents of the table from byte 8 (the end of the metadata length field) all the way to the end of the "Column Names" array. NOTE: It does not include the row count value. See below for an example.
- The size of the "Column Types" and "Column Names" arrays is expected to equal the value stored in "Column Count".

- Column names are limited to the ASCII character set. Strings in row values are still UTF-8 encoded.
- Values with 4-byte (integer) length fields are signed and are limited to a max of 1 megabyte.

Row Data Format:

Each row is prefixed by a 4 byte integer that holds the non-inclusive length of the row. If a row is a single 4-byte integer column, the value of this length prefix will be 4. Row size is artifically restricted to 2 megabytes.

The body of the row is packed array of values. The value at index i is is of type specified by the column type field for index i. The values are serialized according to the serialization rules in "Basic Data Types" above.

Name	**Type**	**Length (bytes)**	**[Basic | Compound | Complex]**			
Row Length	Integer	4	Basic			
Single Row Value Array						

Example VoltTable Serialization:

The following is a 35-byte long serialized table containing one column named "Test" of type BIGINT with one row containing the number 5.

Byte Offset	**Byte Value (dec)**	**Field Desc**	**Field Value**	**Meaning**
00	0	Total Table Length	31	Table size is 31 bytes
01	0			
02	0			
03	31			
04	0	Table Metadata Length	12	Header size is 12 bytes
05	0			
06	0			
07	12			
08	0	Status Code	0	Status code is 0
09	0	Column Count	1	Table contains 1 column
10	1			
11	6	Column 1 Type	VoltType.BIGINT	Column 1 is a BigInt
12	0	Col 1 Name Length	4	The name of column 1 has 4 chars (ASCII)
13	0			
14	0			
15	4			
16	84	Column 1 Name Value	T	Column 1 is named "Test"
17	101		e	
18	115		s	
19	116		t	

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
20	0	Row Count	1	Table has 1 row
21	0			
22	0			
23	1			
24	0	Row 1 Length	8	First row is 8 bytes long
25	0			
26	0			
27	8			
28	0	Row 1 Col 1 Value	5	Value in first row/first col is 5
29	0			
30	0			
31	0			
32	0			
33	0			
34	0			
35	5			

Serializable exceptions

Currently serializable exceptions are not a part of the wire protocol although they are present in the invocation response message. The length every serialized exception (4 byte integer) is part of the invocation response message and it can be used to skip the exception. It is possible to retrieve the Byte ordinal that follows the exception's length preceding value. The ordinal will not be present if the exception's length is 0.

- 1 EEException Generic failure in Volt. Should indicate a failure in the server and not the application code. These should not occur in normal operation.
- 2 SQLException Base class for all exceptions that can occur during normal operation. Things like constraint failures (unique, string length, not null) that are caught and handled correct by Volt.
- 3 ConstraintFailureException Specialization of SQLException for contraint failures during the execution of a stored procedure.

In the future a set of serializable exceptions and their serialization format will be added to the wire protocol.

Parameter Set

A parameter set contains all the parameters to be passed to a stored procedure and it is one of the structures bundled inside a stored procedure invocation request. The first value of a parameter set is a Short indicating the number of parameters that follow. The following values are a series of <wire type, value> pairs. Each value is preceded by its wire type represented as a Byte. NULL is a valid wire type and value and it is not followed by any additional value. Arrays are preceded by the wire type -99 and the array value contains the type of the array elements as well as the number of elements (see Array type). A parameter set cannot contain a nested parameter set (there is no wire type for parameter set).

Note that varbinary values using type number 25 and arrays of bytes using type number -99, followed by type 3 are effectively interchangeable.

Parameter set

Name	Type	Length (bytes)	[Basic | Compound | Complex]			
Parameter count	Short	2	Basic			
Parameters...						

Parameter

Name	Type	Length (bytes)	[Basic | Compound | Complex]
Parameter type	Byte	1	Basic
Parameter	Any Basic | Compound | Complex excluding parameter sets	variable	Basic | Compound | Complex

The following example parameter set serialization shows a parameter set consisting of an array of two strings ("foo1, "foo2") and a decimal.

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
00	0	Parameter Count	2	Parameter set contains two parameters
01	2			
02	-99	Parameter Type	-99	Next parameter is an array
03	9	Element Type	9	Array Elements are Strings
04	0	Element Count	2	Array contains two elements
05	2			
06	0	String Length	4	String is 4 bytes long
07	0			
08	0			
09	4			
10	102	Characters	f	String data is "foo1"
11	111		o	
12	111		o	
13	49		1	
14	0	String Length	4	String is 4 bytes long
15	0			
16	0			
17	4			
18	102	Characters	f	String data is "foo2"
19	111		o	

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
20	111		o	
21	50		2	
22	22	Parameter Type	22	Next parameter is a decimal
23	-1	Decimal data	-23325.23425	Decimal value "-23325.23425"
24	-1			
25	-1			
26	-1			
27	-1			
28	-1			
29	-1			
30	-1			
31	-1			
32	-83			
33	33			
34	-46			
35	-78			
36	57			
37	-39			
38	-128			

Message formats

Message header

The header that is included at the beginning of all messages. The length value includes the protocol version byte but not the 4 byte length value.

Name	Type	Length (bytes)	[Basic | Compound | Complex]
Message length	Integer	4	Basic
Protocol version	Byte	1	Basic

The following table shows and example header for a 140,000 byte message.

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
00	0	Message Length	140000	The message is 140,000 bytes long
01	2			
02	34			
03	-32			
04	0	Protocol version	0	This messages is a Volt Wire Protocol version 0 message

Login message

The login message is the first message a client can send to a server after opening a connection. A client does not need to wait for a response to the login message to begin sending invocation requests. The login message identifies a service to authenticate to. There are currently two supported services: the "database" service authenticates stored procedure callers; the "export" service authenticates export connector callers. Export connectors are extensible and future plugin connectors may handle other service string values.

Name	**Type**	**Length (bytes)**	**[Basic | Compound | Complex]**			
Message Header						
Service	String	variable	Basic			
Username	String	variable	Basic			
SHA-1 password hash	Binary	20	Basic			

The following table shows an example login message for username "scooby" password "doo"

Byte Offset	**Byte Value (dec)**	**Field Desc**	**Field Value**	**Meaning**
00	0	Message Length	43	The message is 42 bytes long
01	0			
02	0			
03	43			
04	0	Protocol version	0	This messages is a Volt Wire Protocol version 0 message
05	0	Service length	8	Service string is 8 bytes long
06	0			
07	0			
08	8			
09		Characters	d	Service string is "database"
10	97		a	
11	116		t	
12	97		a	
13	98		b	
14	97		a	
15	115		s	
16	101		e	
17	0	Username length	6	Username is 6 bytes long
18	0			
19	0			
20	6			
21	115	Characters	s	Username is "scooby"

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
22	99		c	
23	111		o	
24	111		o	
25	98		b	
26	121		y	
27	100	SHA-1 Hash	SHA-1("doo")	Password has is SHA-1("doo")
28	0			
29	-50			
30	-61			
31	125			
32	-52			
33	35			
34	-99			
35	11			
36	-7			
37	-126			
38	-3			
39	108			
40	114			
41	-5			
42	3			
43	-56			
44	-90			
45	-73			
46	-113			

Login response

A response is generated to a login request and success is indicated with a result code of 0. Any other value indicates authentication failure and will be followed by the server closing the connection. A response code of 1 indicates that the there are too many connections. A response code of 2 indicates that authentication failed because the client took too long to transmit credentials. A response code of 3 indicates a corrupt or invalid login message. If the response code is 0 the response will also contain additional information following the result code. A 4 byte integer specifying the host id of the Volt node . An 8 byte long specifying a connection id that is unique among connections to that node. An 8 byte long timestamp (milliseconds since Unix epoch) and a 4 byte IPV4 address representing the time the cluster was started and the address of the leader node. These two values uniquely identify a Volt cluster instance. And finally a string containing a textual description of the build the node being connected to is running.

Name	Type	Length (bytes)	[Basic | Compound | Complex]			
Message Header						

Name	Type	Length (bytes)	[Basic | Compound | Complex]			
Authentication result code	Byte	1	Basic			
Server Host ID	Integer	4	Basic			
Connection ID	Long	8	Basic			
Cluster start timestamp (milliseconds since Unix epoch)	Long	8	Basic			
Leader IPV4 address	Integer	4	Basic			
Build string	String	variable	Basic			

The following table shows a login response indicating success

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
00	0	Message Length	82	The message is 82 bytes long
01	0			
02	0			
03	82			
04	0	Protocol version	0	This messages is a Volt Wire Protocol version 0 message
05	0	Result code	0	Authentication succeded
06	0	Server Host ID	0	The Host ID of the server is 0
07	0			
08	0			
09	0			
10	0	Connection ID	12	The ID of the connection is 12
11	0			
12	0			
13	0			
14	0			
15	0			
16	0			
17	12			
18	0	Cluster start timestamp	105	The cluster was started 105 milliseconds after the Unix epoch
19	0			
20	0			

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
21	0			
22	0			
23	0			
24	0			
25	105			
26	192	Leader IPV4 address	192.168.0.1	The IPV4 address of the leader that started the cluster was 192.168.0.1
27	168			
28	0			
29	1			
30	0	Build string length	52	The length of the build string is 52
31	0			
32	0			
33	52			
34	48 ("0")	Build string	0.7.01 https://svn.voltdb.com/eng/trunk?revision=443	Build is version 0.7.01 off the trunk revision 443/td>
35	46 (".")			
36	55 ("7")			
37	46 (".")			
38	48 ("0")			
39	49 ("1")			
40	32 (" ")			
41	104 ("h"			
42	116 ("t")			
43	116 ("t")			
44	112 ("p")			
45	115 ("s")			
46	58 (":")			
47	47 ("/")			
48	47 ("/")			
49	115 ("s")			
50	118 ("v")			
51	110 ("n")			
52	46 (".")			
53	118 ("v")			
54	111 ("o")			
55	108 ("l")			
56	116 ("t")			
57	100 ("d")			
58	98 ("b")			

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
59	46 (".")			
60	99 ("c")			
61	111 ("o")			
62	109 ("m")			
63	47 ("/")			
64	101 ("e")			
65	110 ("n")			
66	103 ("g")			
67	47 ("/")			
68	116 ("t")			
69	114 ("r")			
70	117 ("u")			
71	110 ("n")			
72	107 ("k")			
73	63 ("?")			
74	114 ("r")			
75	101 ("e")			
76	118 ("v")			
77	105 ("i")			
78	115 ("s")			
79	105 ("i")			
80	111 ("o")			
81	110 ("n")			
82	61 ("=")			
83	52 ("4")			
84	52 ("4")			
85	51 ("3")			

Invocation request

A request to invoke a stored procedure identifies the procedure to invoke by name, the parameters to pass to the procedure, and an 8 byte piece of client data that will be returned with the response to the invocation request. A client does not need to wait for a response to a request to continue sending requests. The server will use TCP backpressure to avoid running out of memory when a client sends too many invocations for the server to handle.

Name	Type	Length (bytes)	[Basic | Compound | Complex]			
Message Header						
Procedure name	String	variable	Basic			

Name	**Type**	**Length (bytes)**	**[Basic | Compound | Complex]**			
Client data	Binary	8	Basic			
Parameters	Parameter Set	variable	Complex			

The following table shows the serialization for invoking a procedure called "proc" with a parameter set containing an array of two strings ("foo1", "foo2") and a decimal value "-23325.23425"

Byte Offset	**Byte Value (dec)**	**Field Desc**	**Field Value**	**Meaning**
00	0	Message Length	56	The message is 56 bytes long
01	0			
02	0			
03	56			
04	0	Protocol version	0	This messages is a Volt Wire Protocol version 0 message
05	0	Procedure name length	4	Procedure name is 4 bytes long
06	0			
07	0			
08	4			
09	112	Characters	p	Stored procedure name is "proc"
10	114		r	
11	111		o	
12	99		c	
13	0	Client Data		Opaque client data
14	1			
15	2			
16	3			
17	4			
18	5			
19	6			
20	7			
21	0	Parameter Count	2	Parameter set contains two parameters
22	2			
23	-99	Parameter Type	-99	Next parameter is an array
24	9	Element Type	9	Array Elements are Strings
25	0	Element Count	2	Array contains two elements
26	2			
27	0	String Length	4	String is 4 bytes long

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
28	0			
29	0			
30	4			
31	102	Characters	f	String data is "foo1"
32	111		o	
33	111		o	
34	49		1	
35	0	String Length	4	String is 4 bytes long
36	0			
37	0			
38	4			
39	102	Characters	f	String data is "foo2"
40	111		o	
41	111		o	
42	50		2	
43	22	Parameter Type	22	Next parameter is a decimal
44	-1	Decimal data	-23325.23425	Decimal value "-23325.23425"
45	-1			
46	-1			
47	-1			
48	-1			
49	-1			
50	-1			
51	-1			
52	-1			
53	-83			
54	33			
55	-46			
56	-78			
57	57			
58	-39			
59	-128			

Invocation response

An invocation response contains the results of the server's attempt to execute the stored procedure. The response includes optional fields and the first byte after the header is used to indicate which optional fields are present. The status string, application status string, and serializable exception are all optional fields. Bit 7 indicates the presence of a serializable exception, bit 6 indicates the presence of a status string, and bit 8 indicates the presence of an app status

string. The serializable exception that can be included in some responses is currently not a part of the wire protocol. The exception length value should be used to skip exceptions if they are present. The status string is used to return any human readable information the server or stored procedure wants to return with the response. The app status code and app status string can be set by application code from within stored procedures and is returned with the response.

Name	**Type**	**Length (bytes)**	**[Basic | Compound | Complex]**			
Message Header						
Client data	Binary	8	Basic			
Fields present	Byte (bit field)	1	Basic			
Status	Byte	1	Basic			
Status string	String (optional field)	variable	Basic			
Application Status	Byte	1	Basic			
Application Status string	String (optional field)	variable	Basic			
Serialized exception length	Integer (optional field)	4	Basic			
Serialized exception	Serializable exception (optional field)	variable	Complex			
Result count	Short	2	Basic			
Result tables	Series of VoltTables	variable	Compound (containing Complex)			

The following table shows a response to a previous invocation. For demonstrational purposes two tables are returned, but in a real failure case there would be no tables returned.

Byte Offset	**Byte Value (dec)**	**Field Desc**	**Field Value**	**Meaning**
00	0	Message Length	109	The message is 109 bytes long
01	0			
02	0			
03	109			
04	0	Protocol version	0	This messages is a Volt Wire Protocol version 0 message
05	0	Client Data		Opaque client data
06	1			
07	2			
08	3			
09	4			
10	5			

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
11	6			
12	7			
13	-32	Fields present	-32	The optional status string, app status string, and serializable exception fields are present
14	2	Status code	2	Status code is graceful failure (2)
15	0	Status String length	4	Status string length is 4
16	0			
17	0			
18	4			
19	102	Status String	f	The status string was "fail"
20	97		a	
21	105		i	
22	108		l	
23	99	Application status code	99	Application status code is 99
24	0	Application status string length	4	Application status string length is 4
25	0			
26	0			
27	4			
28	118	Application status string	v	The application status string was "volt"
29	111		o	
30	108		l	
31	116		t	
32	0	Serialized Exception length	5	Serialized exception is 5 bytes long
33	0			
34	0			
35	5			
36	1	Exception ordinal	1	Exception is a SQL exception
37	0	Exception body	Opaque	Opaque
38	0			
39	0			
40	0			
41	0	Result table count	2	Two result tables follow
42	2			
43	0	Total Table Length	32	Table size is 32 bytes

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
44	0			
45	0			
46	32			
47	0	Table Metadata Length	12	Header size is 12 bytes
48	0			
49	0			
50	12			
51	0	Status Code	0	Status code is 0
52	0	Column Count	1	Table contains 1 column
53	1			
54	6	Column 1 Type	VoltType.BIGINT	Column 1 is a BigInt
55	0	Col 1 Name Length	4	The name of column 1 has 4 chars (ASCII)
56	0			
57	0			
58	4			
59	84	Column 1 Name Value	T	Column 1 is named "Test"
60	101		e	
61	115		s	
62	116		t	
63	0	Row Count	1	Table has 1 row
64	0			
65	0			
66	1			
67	0	Row 1 Length	8	First row is 8 bytes long
68	0			
69	0			
70	8			
71	0	Row 1 Col 1 Value	5	Value in first row/first col 5
72	0			
73	0			
74	0			
75	0			
76	0			
77	0			
78	5			
79	0	Total Table Length	32	Table size is 32 bytes
80	0			
81	0			

Byte Offset	Byte Value (dec)	Field Desc	Field Value	Meaning
82	32			
83	0	Table Metadata Length	12	Header size is 12 bytes
84	0			
85	0			
86	12			
87	0	Status Code	0	Status code is 0
88	0	Column Count	1	Table contains 1 column
89	1			
90	6	Column 1 Type	VoltType.BIGINT	Column 1 is a BigInt
91	0	Col 1 Name Length	4	The name of column 1 has 4 chars (ASCII)
92	0			
93	0			
94	4			
95	84	Column 1 Name Value	T	Column 1 is named "Test"
96	101		e	
97	115		s	
98	116		t	
99	0	Row Count	1	Table has 1 row
100	0			
101	0			
102	1			
103	0	Row 1 Length	8	First row is 8 bytes long
104	0			
105	0			
106	8			
107	0	Row 1 Col 1 Value	5	Value in first row/first col 5
108	0			
109	0			
110	0			
111	0			
112	0			
113	0			
114	5			