Infinispan Hot Rod Protocol Reference

Table of Contents

Chapter 1. Hot Rod Protocol

The following articles provides detailed information about each version of the custom TCP client/server Hot Rod protocol.

- Hot Rod Protocol 1.0 (Infinispan 4.1)
- Hot Rod Protocol 1.1 (Infinispan 5.1)
- Hot Rod Protocol 1.2 (Infinispan 5.2)
- Hot Rod Protocol 1.3 (Infinispan 6.0)
- Hot Rod Protocol 2.0 (Infinispan 7.0)
- Hot Rod Protocol 2.1 (Infinispan 7.1)
- Hot Rod Protocol 2.2 (Infinispan 8.0)
- Hot Rod Protocol 2.3 (Infinispan 8.0)
- Hot Rod Protocol 2.4 (Infinispan 8.1)
- Hot Rod Protocol 2.5 (Infinispan 8.2)
- Hot Rod Protocol 2.6 (Infinispan 9.0)
- Hot Rod Protocol 2.7 (Infinispan 9.2)
- Hot Rod Protocol 2.8 (Infinispan 9.3)
- Hot Rod Protocol 2.9 (Infinispan 9.4)
- Hot Rod Protocol 3.0 (Infinispan 10.0)

1.1. Hot Rod Protocol 1.0

Infinispan versions

This version of the protocol was implemented since Infinispan 4.1.0.Final and is no longer supported since Infinispan 10.

All key and values are sent and stored as byte arrays. Hot Rod makes no assumptions about their types.

Some clarifications about the other types:

- vInt : Variable-length integers are defined defined as compressed, positive integers where the high-order bit of each byte indicates whether more bytes need to be read. The low-order seven bits are appended as increasingly more significant bits in the resulting integer value making it efficient to decode. Hence, values from zero to 127 are stored in a single byte, values from 128 to 16,383 are stored in two bytes, and so on:

Value	First byte	Second byte	Third byte
0	00000000		
1	00000001		

Value	First byte	Second byte	Third byte
2	00000010		
…			
127	01111111		
128	10000000	00000001	
129	10000001	00000001	
130	10000010	00000001	
…			
16,383	11111111	01111111	
16,384	10000000	10000000	00000001
16,385	10000001	10000000	00000001
…			

- signed vInt: The vInt above is also able to encode negative values, but will always use the maximum size (5 bytes) no matter how small the endoded value is. In order to have a small payload for negative values too, signed vInts uses ZigZag encoding on top of the vInt encoding. More details here
- vLong : Refers to unsigned variable length long values similar to vInt but applied to longer values. They're between 1 and 9 bytes long.
- String : Strings are always represented using UTF-8 encoding.

1.1.1. Request Header

The header for a request is composed of:

Table 1. Request header

Field Name	Size	Value
Magic	1 byte	0xA0 = request
Message ID	vLong	ID of the message that will be copied back in the response. This allows for Hot Rod clients to implement the protocol in an asynchronous way.
Version	1 byte	Hot Rod server version. In this particular case, this is 10

Field Name	Size	Value
Opcode	1 byte	Request operation code: 0x01 = put (since 1.0) 0x03 = get (since 1.0) 0x05 = putIfAbsent (since 1.0) 0x07 = replace (since 1.0) 0x09 = replaceIfUnmodified (since 1.0) 0x0B = remove (since 1.0) 0x0D = removeIfUnmodified (since 1.0) 0x0F = containsKey (since 1.0) 0x11 = getWithVersion (since 1.0) 0x13 = clear (since 1.0) 0x15 = stats (since 1.0) 0x17 = ping (since 1.0) 0x19 = bulkGet (since 1.2) 0x1B = getWithMetadata (since 1.2) 0x1D = bulkGetKeys (since 1.2) 0x1F = query (since 1.3) 0x21 = authMechList (since 2.0) 0x23 = auth (since 2.0) 0x25 = addClientListener (since 2.0) 0x27 = removeClientListener (since 2.0) 0x29 = size (since 2.0) 0x2B = exec (since 2.1) 0x2D = putAll (since 2.1) 0x2F = getAll (since 2.1) 0x31 = iterationStart (since 2.3) 0x33 = iterationNext (since 2.3) 0x35 = iterationEnd (since 2.3) 0x37 = getStream (since 2.6) 0x39 = putStream (since 2.6)
Cache Name Length	vInt	Length of cache name. If the passed length is 0 (followed by no cache name), the operation will interact with the default cache.
Cache Name	string	Name of cache on which to operate. This name must match the name of predefined cache in the Infinispan configuration file.
Flags	vInt	A variable length number representing flags passed to the system. Each flags is represented by a bit. Note that since this field is sent as variable length, the most significant bit in a byte is used to determine whether more bytes need to be read, hence this bit does not represent any flag. Using this model allows for flags to be combined in a short space. Here are the current values for each flag: 0x0001 = force return previous value

Field Name	Size	Value
Client Intelligence	1 byte	This byte hints the server on the client capabilities: 0x01 = basic client, interested in neither cluster nor hash information 0x02 = topology-aware client, interested in cluster information 0x03 = hash-distribution-aware client, that is interested in both cluster and hash information
Topology Id	vInt	This field represents the last known view in the client. Basic clients will only send 0 in this field. When topology-aware or hash-distribution-aware clients will send 0 until they have received a reply from the server with the current view id. Afterwards, they should send that view id until they receive a new view id in a response.
Transaction Type	1 byte	This is a 1 byte field, containing one of the following well-known supported transaction types (For this version of the protocol, the only supported transaction type is 0): 0 = Non-transactional call, or client does not support transactions. The subsequent TX_ID field will be omitted. 1 = X/Open XA transaction ID (XID). This is a well-known, fixed-size format.
Transaction Id	byte array	The byte array uniquely identifying the transaction associated to this call. Its length is determined by the transaction type. If transaction type is 0, no transaction id will be present.

1.1.2. Response Header

The header for a response is composed of:

Table 2. Response header

Field Name	Size	Value
Magic	1 byte	0xA1 = response
Message ID	vLong	ID of the message, matching the request for which the response is sent.

Field Name	Size	Value
Opcode	1 byte	Response operation code: 0x02 = put (since 1.0) 0x04 = get (since 1.0) 0x06 = putIfAbsent (since 1.0) 0x08 = replace (since 1.0) 0x0A = replaceIfUnmodified (since 1.0) 0x0C = remove (since 1.0) 0x0E = removeIfUnmodified (since 1.0) 0x10 = containsKey (since 1.0) 0x12 = getWithVersion (since 1.0) 0x14 = clear (since 1.0) 0x16 = stats (since 1.0) 0x18 = ping (since 1.0) 0x1A = bulkGet (since 1.0) 0x1C = getWithMetadata (since 1.2) 0x1E = bulkGetKeys (since 1.2) 0x20 = query (since 1.3) 0x22 = authMechList (since 2.0) 0x24 = auth (since 2.0) 0x26 = addClientListener (since 2.0) 0x28 = removeClientListener (since 2.0) 0x2A = size (since 2.0) 0x2C = exec (since 2.1) 0x2E = putAll (since 2.1) 0x30 = getAll (since 2.1) 0x32 = iterationStart (since 2.3) 0x34 = iterationNext (since 2.3) 0x36 = iterationEnd (since 2.3) 0x38 = getStream (since 2.6) 0x3A = putStream (since 2.6) 0x50 = error (since 1.0)
Status	1 byte	Status of the response, possible values: 0x00 = No error 0x01 = Not put/removed/replaced 0x02 = Key does not exist 0x81 = Invalid magic or message id 0x82 = Unknown command 0x83 = Unknown version 0x84 = Request parsing error 0x85 = Server Error 0x86 = Command timed out
Topology Change Marker	string	This is a marker byte that indicates whether the response is prepended with topology change information. When no topology change follows, the content of this byte is 0. If a topology change follows, its contents are 1.

Exceptional error status responses, those that start with 0x8 ..., are followed by the length of the error message (as a vInt) and error message itself as String.

1.1.3. Topology Change Headers

The following section discusses how the response headers look for topology-aware or hash-distribution-aware clients when there's been a cluster or view formation change. Note that it's the server that makes the decision on whether it sends back the new topology based on the current topology id and the one the client sent. If they're different, it will send back the new topology.

1.1.4. Topology-Aware Client Topology Change Header

This is what topology-aware clients receive as response header when a topology change is sent back:

Field Name	Size	Value
Response header with topology change marker	variable	See previous section.
Topology Id	vInt	Topology ID
Num servers in topology	vInt	Number of Hot Rod servers running within the cluster. This could be a subset of the entire cluster if only a fraction of those nodes are running Hot Rod servers.
m1: Host/IP length	vInt	Length of hostname or IP address of individual cluster member that Hot Rod client can use to access it. Using variable length here allows for covering for hostnames, IPv4 and IPv6 addresses.
m1: Host/IP address	string	String containing hostname or IP address of individual cluster member that Hot Rod client can use to access it.
m1: Port	2 bytes (Unsigned Short)	Port that Hot Rod clients can use to communicate with this cluster member.
m2: Host/IP length	vInt	
m2: Host/IP address	string	
m2: Port	2 bytes (Unsigned Short)	
...etc		

1.1.5. Distribution-Aware Client Topology Change Header

This is what hash-distribution-aware clients receive as response header when a topology change is sent back:

Field Name	Size	Value
Response header with topology change marker	variable	See previous section.
Topology Id	vInt	Topology ID
Num Key Owners	2 bytes (Unsigned Short)	Globally configured number of copies for each Infinispan distributed key
Hash Function Version	1 byte	Hash function version, pointing to a specific hash function in use. See Hot Rod hash functions for details.
Hash space size	vInt	Modulus used by Infinispan for for all module arithmetic related to hash code generation. Clients will likely require this information in order to apply the correct hash calculation to the keys.
Num servers in topology	vInt	Number of Infinispan Hot Rod servers running within the cluster. This could be a subset of the entire cluster if only a fraction of those nodes are running Hot Rod servers.
m1: Host/IP length	vInt	Length of hostname or IP address of individual cluster member that Hot Rod client can use to access it. Using variable length here allows for covering for hostnames, IPv4 and IPv6 addresses.
m1: Host/IP address	string	String containing hostname or IP address of individual cluster member that Hot Rod client can use to access it.
m1: Port	2 bytes (Unsigned Short)	Port that Hot Rod clients can use to communicat with this cluster member.
m1: Hashcode	4 bytes	32 bit integer representing the hashcode of a cluster member that a Hot Rod client can use indentify in which cluster member a key is located having applied the CSA to it.
m2: Host/IP length	vInt	
m2: Host/IP address	string	
m2: Port	2 bytes (Unsigned Short)	
m2: Hashcode	4 bytes	
...etc		

It's important to note that since hash headers rely on the consistent hash algorithm used by the server and this is a factor of the cache interacted with, hash-distribution-aware headers can only be returned to operations that target a particular cache. Currently ping command does not target any cache (this is to change as per ISPN-424) , hence calls to ping command with hash-topology-aware client settings will return a hash-distribution-aware header with "Num Key Owners", "Hash

Function Version", "Hash space size" and each individual host's hash code all set to 0. This type of header will also be returned as response to operations with hash-topology-aware client settings that are targeting caches that are not configured with distribution.

1.1.6. Operations

Get (0x03)/Remove (0x0B)/ContainsKey (0x0F)/GetWithVersion (0x11)

Common request format:

Field Name	Size	Value
Header	variable	Request header
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.

Get response (0x04):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if key retrieved 0x02 = if key does not exist
Value Length	vInt	If success, length of value
Value	byte array	If success, the requested value

Remove response (0x0C):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if key removed 0x02 = if key does not exist
Previous value Length	vInt	If force return previous value flag was sent in the request and the key was removed, the length of the previous value will be returned. If the key does not exist, value length would be 0. If no flag was sent, no value length would be present.
Previous value	byte array	If force return previous value flag was sent in the request and the key was removed, previous value.

ContainsKey response (0x10):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if key exists 0x02 = if key does not exist

GetWithVersion response (0x12):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if key retrieved 0x02 = if key does not exist
Entry Version	8 bytes	Unique value of an existing entry's modification. The protocol does not mandate that entry_version values are sequential. They just need to be unique per update at the key level.
Value Length	vInt	If success, length of value
Value	byte array	If success, the requested value

BulkGet

Request (0x19):

Field Name	Size	Value
Header	variable	Request header
Entry count	vInt	Maximum number of Infinispan entries to be returned by the server (entry == key + associated value). Needed to support CacheLoader.load(int). If 0 then all entries are returned (needed for CacheLoader.loadAll()).

Response (0x20):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, data follows
More	1 byte	One byte representing whether more entries need to be read from the stream. So, when it's set to 1, it means that an entry follows, whereas when it's set to 0, it's the end of stream and no more entries are left to read. For more information on BulkGet look here
Key 1 Length	vInt	Length of key
Key 1	byte array	Retrieved key
Value 1 Length	vInt	Length of value
Value 1	byte array	Retrieved value

Field Name	Size	Value
More	1 byte	
Key 2 Length	vInt	
Key 2	byte array	
Value 2 Length	vInt	
Value 2	byte array	
… etc		

Put (0x01)/PutIfAbsent (0x05)/Replace (0x07)

Common request format:

Field Name	Size	Value
Header	variable	Request header
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.
Lifespan	vInt	Number of seconds that a entry during which the entry is allowed to life. If number of seconds is bigger than 30 days, this number of seconds is treated as UNIX time and so, represents the number of seconds since 1/1/1970. If set to 0, lifespan is unlimited.
Max Idle	vInt	Number of seconds that a entry can be idle before it's evicted from the cache. If 0, no max idle time.
Value Length	vInt	Length of value
Value	byte-array	Value to be stored

Put response (0x02):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if stored
Previous value Length	vInt	If force return previous value flag was sent in the request and the key was put, the length of the previous value will be returned. If the key does not exist, value length would be 0. If no flag was sent, no value length would be present.
Previous value	byte array	If force return previous value flag was sent in the request and the key was put, previous value.

Replace response (0x08):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if stored 0x01 = if store did not happen because key does not exist
Previous value Length	vInt	If force return previous value flag was sent in the request, the length of the previous value will be returned. If the key does not exist, value length would be 0. If no flag was sent, no value length would be present.
Previous value	byte array	If force return previous value flag was sent in the request and the key was replaced, previous value.

PutIfAbsent response (0x06):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if stored 0x01 = if store did not happen because key was present
Previous value Length	vInt	If force return previous value flag was sent in the request, the length of the previous value will be returned. If the key does not exist, value length would be 0. If no flag was sent, no value length would be present.
Previous value	byte array	If force return previous value flag was sent in the request and the key was replaced, previous value.

ReplaceIfUnmodified

Request (0x09):

Field Name	Size	Value
Header	variable	Request header
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.
Lifespan	vInt	Number of seconds that a entry during which the entry is allowed to life. If number of seconds is bigger than 30 days, this number of seconds is treated as UNIX time and so, represents the number of seconds since 1/1/1970. If set to 0, lifespan is unlimited.

Field Name	Size	Value
Max Idle	vInt	Number of seconds that a entry can be idle before it's evicted from the cache. If 0, no max idle time.
Entry Version	8 bytes	Use the value returned by GetWithVersion operation.
Value Length	vInt	Length of value
Value	byte-array	Value to be stored

Response (0x0A):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if replaced 0x01 = if replace did not happen because key had been modified 0x02 = if not replaced because if key does not exist
Previous value Length	vInt	If force return previous value flag was sent in the request, the length of the previous value will be returned. If the key does not exist, value length would be 0. If no flag was sent, no value length would be present.
Previous value	byte array	If force return previous value flag was sent in the request and the key was replaced, previous value.

RemoveIfUnmodified

Request (0x0D):

Field Name	Size	Value
Header	variable	Request header
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.
Entry Version	8 bytes	Use the value returned by GetWithMetadata operation.

Response (0x0E):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if removed 0x01 = if remove did not happen because key had been modified 0x02 = if not removed because key does not exist

Field Name	Size	Value
Previous value Length	vInt	If force return previous value flag was sent in the request, the length of the previous value will be returned. If the key does not exist, value length would be 0. If no flag was sent, no value length would be present.
Previous value	byte array	If force return previous value flag was sent in the request and the key was removed, previous value.

Clear

Request (0x13):

Field Name	Size	Value
Header	variable	Request header

Response (0x14):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if cleared

PutAll

Bulk operation to put all key value entries into the cache at the same time.

Request (0x2D):

Field Name	Size	Value
Header	variable	Request header
Lifespan	vInt	Number of seconds that provided entries are allowed to live. If number of seconds is bigger than 30 days, this number of seconds is treated as UNIX time and so, represents the number of seconds since 1/1/1970. If set to 0, lifespan is unlimited.
Max Idle	vInt	Number of seconds that each entry can be idle before it's evicted from the cache. If 0, no max idle time.
Entry count	vInt	How many entries are being inserted
Key 1 Length	vInt	Length of key
Key 1	byte array	Retrieved key
Value 1 Length	vInt	Length of value
Value 1	byte array	Retrieved value
Key 2 Length	vInt	
Key 2	byte array	
Value 2 Length	vInt	

Field Name	Size	Value
Value 2	byte array	
… continues until entry count is reached		

Response (0x2E):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if all put

GetAll

Bulk operation to get all entries that map to a given set of keys.

Request (0x2F):

Field Name	Size	Value
Header	variable	Request header
Key count	vInt	How many keys to find entries for
Key 1 Length	vInt	Length of key
Key 1	byte array	Retrieved key
Key 2 Length	vInt	
Key 2	byte array	
… continues until key count is reached		

Response (0x30):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	
Entry count	vInt	How many entries are being returned
Key 1 Length	vInt	Length of key
Key 1	byte array	Retrieved key
Value 1 Length	vInt	Length of value
Value 1	byte array	Retrieved value
Key 2 Length	vInt	

Field Name	Size	Value
Key 2	byte array	
Value 2 Length	vInt	
Value 2	byte array	
… continues until entry count is reached		0x00 = success, if the get returned sucessfully

Stats

Returns a summary of all available statistics. For each statistic returned, a name and a value is returned both in String UTF-8 format. The supported stats are the following:

Name	Explanation
timeSinceStart	Number of seconds since Hot Rod started.
currentNumberOfEntries	Number of entries currently in the Hot Rod server.
totalNumberOfEntries	Number of entries stored in Hot Rod server.
stores	Number of put operations.
retrievals	Number of get operations.
hits	Number of get hits.
misses	Number of get misses.
removeHits	Number of removal hits.
removeMisses	Number of removal misses.

Request (0x15):

Field Name	Size	Value
Header	variable	Request header

Response (0x16):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if stats retrieved
Number of stats	vInt	Number of individual stats returned.
Name 1 length	vInt	Length of named statistic.
Name 1	string	String containing statistic name.
Value 1 length	vInt	Length of value field.
Value 1	string	String containing statistic value.

Field Name	Size	Value
Name 2 length	vInt	
Name 2	string	
Value 2 length	vInt	
Value 2	String	
…etc		

Ping

Application level request to see if the server is available.

Request (0x17):

Field Name	Size	Value
Header	variable	Request header

Response (0x18):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if no errors

Error Handling

Error response (0x50)

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x8x = error response code
Error Message Length	vInt	Length of error message
Error Message	string	Error message. In the case of 0x84 , this error field contains the latest version supported by the Hot Rod server. Length is defined by total body length.

Multi-Get Operations

A multi-get operation is a form of get operation that instead of requesting a single key, requests a set of keys. The Hot Rod protocol does not include such operation but remote Hot Rod clients could easily implement this type of operations by either parallelizing/pipelining individual get requests. Another possibility would be for remote clients to use async or non-blocking get requests. For example, if a client wants N keys, it could send send N async get requests and then wait for all the replies. Finally, multi-get is not to be confused with bulk-get operations. In bulk-gets, either all or a number of keys are retrieved, but the client does not know which keys to retrieve, whereas in multi-get, the client defines which keys to retrieve.

1.1.7. Example - Put request

- Coded request

Byte	0	1	2	3	4	5	6	7
8	0xA0	0x09	0x41	0x01	0x07	0x4D ('M')	0x79 ('y')	0x43 ('C')
16	0x61 ('a')	0x63 ('c')	0x68 ('h')	0x65 ('e')	0x00	0x03	0x00	0x00
24	0x00	0x05	0x48 ('H')	0x65 ('e')	0x6C ('l')	0x6C ('l')	0x6F ('o')	0x00
32	0x00	0x05	0x57 ('W')	0x6F ('o')	0x72 ('r')	0x6C ('l')	0x64 ('d')	

- Field explanation

Field Name	Value	Field Name	Value
Magic (0)	0xA0	Message Id (1)	0x09
Version (2)	0x41	Opcode (3)	0x01
Cache name length (4)	0x07	Cache name(5-11)	'MyCache'
Flag (12)	0x00	Client Intelligence (13)	0x03
Topology Id (14)	0x00	Transaction Type (15)	0x00
Transaction Id (16)	0x00	Key field length (17)	0x05
Key (18 - 22)	'Hello'	Lifespan (23)	0x00
Max idle (24)	0x00	Value field length (25)	0x05
Value (26-30)	'World'		

- Coded response

Byte	0	1	2	3	4	5	6	7
8	0xA1	0x09	0x01	0x00	0x00			

- Field Explanation

Field Name	Value	Field Name	Value
Magic (0)	0xA1	Message Id (1)	0x09
Opcode (2)	0x01	Status (3)	0x00
Topology change marker (4)	0x00		

1.2. Hot Rod Protocol 1.1

Infinispan versions

This version of the protocol was implemented since Infinispan 5.1.0.FINAL and is no longer supported since Infinispan 10.

1.2.1. Request Header

The `version` field in the header is updated to 11.

1.2.2. Distribution-Aware Client Topology Change Header

Updated for 1.1

This section has been modified to be more efficient when talking to distributed caches with virtual nodes enabled.

This is what hash-distribution-aware clients receive as response header when a topology change is sent back:

Field Name	Size	Value
Response header with topology change marker	variable	See previous section.
Topology Id	vInt	Topology ID
Num Key Owners	2 bytes (Unsigned Short)	Globally configured number of copies for each Infinispan distributed key
Hash Function Version	1 byte	Hash function version, pointing to a specific hash function in use. See Hot Rod hash functions for details.
Hash space size	vInt	Modulus used by Infinispan for for all module arithmetic related to hash code generation. Clients will likely require this information in order to apply the correct hash calculation to the keys.
Num servers in topology	vInt	Number of Hot Rod servers running within the cluster. This could be a subset of the entire cluster if only a fraction of those nodes are running Hot Rod servers.
Num Virtual Nodes Owners	vInt	Field added in version 1.1 of the protocol that represents the number of configured virtual nodes. If no virtual nodes are configured or the cache is not configured with distribution, this field will contain 0.
m1: Host/IP length	vInt	Length of hostname or IP address of individual cluster member that Hot Rod client can use to access it. Using variable length here allows for covering for hostnames, IPv4 and IPv6 addresses.
m1: Host/IP address	string	String containing hostname or IP address of individual cluster member that Hot Rod client can use to access it.

Field Name	Size	Value
m1: Port	2 bytes (Unsigned Short)	Port that Hot Rod clients can use to communicat with this cluster member.
m1: Hashcode	4 bytes	32 bit integer representing the hashcode of a cluster member that a Hot Rod client can use indentify in which cluster member a key is located having applied the CSA to it.
m2: Host/IP length	vInt	
m2: Host/IP address	string	
m2: Port	2 bytes (Unsigned Short)	
m2: Hashcode	4 bytes	
...etc		

1.2.3. Server node hash code calculation

Adding support for virtual nodes has made version 1.0 of the Hot Rod protocol impractical due to bandwidth it would have taken to return hash codes for all virtual nodes in the clusters (this number could easily be in the millions). So, as of version 1.1 of the Hot Rod protocol, clients are given the base hash id or hash code of each server, and then they have to calculate the real hash position of each server both with and without virtual nodes configured. Here are the rules clients should follow when trying to calculate a node's hash code:

1\. With *virtual nodes disabled* : Once clients have received the base hash code of the server, they need to normalize it in order to find the exact position of the hash wheel. The process of normalization involves passing the base hash code to the hash function, and then do a small calculation to avoid negative values. The resulting number is the node's position in the hash wheel:

```
public static int getNormalizedHash(int nodeBaseHashCode, Hash hashFct) {
   return hashFct.hash(nodeBaseHashCode) & Integer.MAX_VALUE; // make sure no negative
numbers are involved.
}
```

2\. With *virtual nodes enabled* : In this case, each node represents N different virtual nodes, and to calculate each virtual node's hash code, we need to take the the range of numbers between 0 and N-1 and apply the following logic:

- For virtual node with 0 as id, use the technique used to retrieve a node's hash code, as shown in the previous section.
- For virtual nodes from 1 to N-1 ids, execute the following logic:

```
public static int virtualNodeHashCode(int nodeBaseHashCode, int id, Hash hashFct) {
   int virtualNodeBaseHashCode = id;
   virtualNodeBaseHashCode = 31 * virtualNodeBaseHashCode + nodeBaseHashCode;
   return getNormalizedHash(virtualNodeBaseHashCode, hashFct);
}
```

1.3. Hot Rod Protocol 1.2

Infinispan versions

This version of the protocol was implemented since Infinispan 5.2.0.Final and is no longer supported since Infinispan 10.

Since Infinispan 5.3.0, HotRod supports encryption via SSL. However, since this only affects the transport, the version number of the protocol has not been incremented.

1.3.1. Request Header

The `version` field in the header is updated to `12`.

Two new request operation codes have been added:

- 0x1B = getWithMetadata request
- 0x1D = bulkKeysGet request

Two new flags have been added too:

- 0x0002 = use cache-level configured default lifespan
- 0x0004 = use cache-level configured default max idle

1.3.2. Response Header

Two new response operation codes have been added:

- 0x1C = getWithMetadata response
- 0x1E = bulkKeysGet response

1.3.3. Operations

GetWithMetadata

Request (0x1B):

Field Name	Size	Value
Header	variable	Request header

Field Name	Size	Value
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.

Response (0x1C):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if key retrieved 0x02 = if key does not exist
Flag	1 byte	A flag indicating whether the response contains expiration information. The value of the flag is obtained as a bitwise OR operation between INFINITE_LIFESPAN (0x01) and `INFINITE_MAXIDLE (0x02)`.
Created	Long	(optional) a Long representing the timestamp when the entry was created on the server. This value is returned only if the flag's INFINITE_LIFESPAN bit is not set.
Lifespan	vInt	(optional) a vInt representing the lifespan of the entry in seconds. This value is returned only if the flag's INFINITE_LIFESPAN bit is not set.
LastUsed	Long	(optional) a Long representing the timestamp when the entry was last accessed on the server. This value is returned only if the flag's `INFINITE_MAXIDLE` bit is not set.
MaxIdle	vInt	(optional) a vInt representing the maxIdle of the entry in seconds. This value is returned only if the flag's `INFINITE_MAXIDLE` bit is not set.
Entry Version	8 bytes	Unique value of an existing entry's modification. The protocol does not mandate that entry_version values are sequential. They just need to be unique per update at the key level.
Value Length	vInt	If success, length of value
Value	byte array	If success, the requested value

BulkKeysGet

Request (0x1D):

Field Name	Size	Value
Header	variable	Request header

Field Name	Size	Value
Scope	vInt	0 = Default Scope - This scope is used by RemoteCache.keySet() method. If the remote cache is a distributed cache, the server launch a stream operation to retrieve all keys from all of the nodes. (Remember, a topology-aware Hot Rod Client could be load balancing the request to any one node in the cluster). Otherwise, it'll get keys from the cache instance local to the server receiving the request (that is because the keys should be the same across all nodes in a replicated cache). 1 = Global Scope - This scope behaves the same to Default Scope. 2 = Local Scope - In case when remote cache is a distributed cache, the server will not launch a stream operation to retrieve keys from all nodes. Instead, it'll only get keys local from the cache instance local to the server receiving the request.

Response (0x1E):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, data follows
More	1 byte	One byte representing whether more keys need to be read from the stream. So, when it's set to 1, it means that an entry follows, whereas when it's set to 0, it's the end of stream and no more entries are left to read. For more information on BulkGet look here
Key 1 Length	vInt	Length of key
Key 1	byte array	Retrieved key
More	1 byte	
Key 2 Length	vInt	
Key 2	byte array	
... etc		

1.4. Hot Rod Protocol 1.3

Infinispan versions

This version of the protocol was implemented since Infinispan 6.0.0.Final and is no longer supported since Infinispan 10.

1.4.1. Request Header

The `version` field in the header is updated to `13`.

A new request operation code has been added:

- 0x1F = query request

1.4.2. Response Header

A new response operation code has been added:

- 0x20 = query response

1.4.3. Operations

Query

Request (0x1F):

Field Name	Size	Value
Header	variable	Request header
Query Length	vInt	The length of the protobuf encoded query object
Query	byte array	Byte array containing the protobuf encoded query object, having a length specified by previous field.

Response (0x20):

Field Name	Size	Value
Header	variable	Response header
Response payload Length	vInt	The length of the protobuf encoded response object
Response payload	byte array	Byte array containing the protobuf encoded response object, having a length specified by previous field.

As of Infinispan 6.0, the query and response objects are specified by the protobuf message types 'org.infinispan.client.hotrod.impl.query.QueryRequest' and 'org.infinispan.client.hotrod.impl.query.QueryResponse' defined in remote-query/remote-query-client/src/main/resources/org/infinispan/query/remote/client/query.proto. These definitions could change in future Infinispan versions, but as long as these evolutions will be kept backward compatible (according to the rules defined here) no new Hot Rod protocol version will be introduced to accommodate this.

1.5. Hot Rod Protocol 2.0

Infinispan versions

This version of the protocol is implemented since Infinispan 7.0.0.Final.

1.5.1. Request Header

The request header no longer contains `Transaction Type` and `Transaction ID` elements since they're not in use, and even if they were in use, there are several operations for which they would not

make sense, such as `ping` or `stats` commands. Once transactions are implemented, the protocol version will be upped, with the necessary changes in the request header.

The `version` field in the header is updated to `20`.

Two new flags have been added:

- 0x0008 = operation skips loading from configured cache loader.
- 0x0010 = operation skips indexing. Only relevant when the query module is enabled for the cache

The following new request operation codes have been added:

- 0x21 = auth mech list request
- 0x23 = auth request
- 0x25 = add client remote event listener request
- 0x27 = remove client remote event listener request
- 0x29 = size request

1.5.2. Response Header

The following new response operation codes have been added:

- 0x22 = auth mech list response
- 0x24 = auth mech response
- 0x26 = add client remote event listener response
- 0x28 = remove client remote event listener response
- 0x2A = size response

Two new error codes have also been added to enable clients more intelligent decisions, particularly when it comes to fail-over logic:

- 0x87 = Node suspected. When a client receives this error as response, it means that the node that responded had an issue sending an operation to a third node, which was suspected. Generally, requests that return this error should be failed-over to other nodes.
- 0x88 = Illegal lifecycle state. When a client receives this error as response, it means that the server-side cache or cache manager are not available for requests because either stopped, they're stopping or similar situation. Generally, requests that return this error should be failed-over to other nodes.

Some adjustments have been made to the responses for the following commands in order to better handle response decoding without the need to keep track of the information sent. More precisely, the way previous values are parsed has changed so that the status of the command response provides clues on whether the previous value follows or not. More precisely:

- Put response returns `0x03` status code when put was successful and previous value follows.

- PutIfAbsent response returns `0x04` status code only when the putIfAbsent operation failed because the key was present and its value follows in the response. If the putIfAbsent worked, there would have not been a previous value, and hence it does not make sense returning anything extra.
- Replace response returns `0x03` status code only when replace happened and the previous or replaced value follows in the response. If the replace did not happen, it means that the cache entry was not present, and hence there's no previous value that can be returned.
- ReplaceIfUnmodified returns `0x03` status code only when replace happened and the previous or replaced value follows in the response.
- ReplaceIfUnmodified returns `0x04` status code only when replace did not happen as a result of the key being modified, and the modified value follows in the response.
- Remove returns `0x03` status code when the remove happened and the previous or removed value follows in the response. If the remove did not occur as a result of the key not being present, it does not make sense sending any previous value information.
- RemoveIfUnmodified returns `0x03` status code only when remove happened and the previous or replaced value follows in the response.
- RemoveIfUnmodified returns `0x04` status code only when remove did not happen as a result of the key being modified, and the modified value follows in the response.

1.5.3. Distribution-Aware Client Topology Change Header

In Infinispan 5.2, virtual nodes based consistent hashing was abandoned and instead segment based consistent hash was implemented. In order to satisfy the ability for Hot Rod clients to find data as reliably as possible, Infinispan has been transforming the segment based consistent hash to fit Hot Rod 1.x protocol. Starting with version 2.0, a brand new distribution-aware topology change header has been implemented which suppors segment based consistent hashing suitably and provides 100% data location guarantees.

Field Name	Size	Value
Response header with topology change marker	variable	
Topology Id	vInt	Topology ID
Num servers in topology	vInt	Number of Infinispan Hot Rod servers running within the cluster. This could be a subset of the entire cluster if only a fraction of those nodes are running Hot Rod servers.
m1: Host/IP length	vInt	Length of hostname or IP address of individual cluster member that Hot Rod client can use to access it. Using variable length here allows for covering for hostnames, IPv4 and IPv6 addresses.
m1: Host/IP address	string	String containing hostname or IP address of individual cluster member that Hot Rod client can use to access it.

Field Name	Size	Value
m1: Port	2 bytes (Unsigned Short)	Port that Hot Rod clients can use to communicat with this cluster member.
m2: Host/IP length	vInt	
m2: Host/IP address	string	
m2: Port	2 bytes (Unsigned Short)	
…	…	
Hash Function Version	1 byte	Hash function version, pointing to a specific hash function in use. See Hot Rod hash functions for details.
Num segments in topology	vInt	Total number of segments in the topology
Number of owners in segment	1 byte	This can be either 0, 1 or 2 owners.
First owner's index	vInt	Given the list of all nodes, the position of this owner in this list. This is only present if number of owners for this segment is 1 or 2.
Second owner's index	vInt	Given the list of all nodes, the position of this owner in this list. This is only present if number of owners for this segment is 2.

Given this information, Hot Rod clients should be able to recalculate all the hash segments and be able to find out which nodes are owners for each segment. Even though there could be more than 2 owners per segment, Hot Rod protocol limits the number of owners to send for efficiency reasons.

1.5.4. Operations

Auth Mech List

Request (0x21):

Field Name	Size	Value
Header	variable	Request header

Response (0x22):

Field Name	Size	Value
Header	variable	Response header
Mech count	vInt	The number of mechs

Field Name	Size	Value
Mech 1	string	String containing the name of the SASL mech in its IANA-registered form (e.g. GSSAPI, CRAM-MD5, etc)
Mech 2	string	
…etc		

The purpose of this operation is to obtain the list of valid SASL authentication mechs supported by the server. The client will then need to issue an Authenticate request with the preferred mech.

Authenticate

Request (0x23):

Field Name	Size	Value
Header	variable	Request header
Mech	string	String containing the name of the mech chosen by the client for authentication. Empty on the successive invocations
Response length	vInt	Length of the SASL client response
Response data	byte array	The SASL client response

Response (0x24):

Field Name	Size	Value
Header	variable	Response header
Completed	byte	0 if further processing is needed, 1 if authentication is complete
Challenge length	vInt	Length of the SASL server challenge
Challenge data	byte array	The SASL server challenge

The purpose of this operation is to authenticate a client against a server using SASL. The authentication process, depending on the chosen mech, might be a multi-step operation. Once complete the connection becomes authenticated

Add client listener for remote events

Request (0x25):

Field Name	Size	Value
Header	variable	Request header
Listener ID	byte array	Listener identifier

Field Name	Size	Value
Include state	byte	When this byte is set to 1, cached state is sent back to remote clients when either adding a cache listener for the first time, or when the node where a remote listener is registered changes in a clustered environment. When enabled, state is sent back as cache entry created events to the clients. If set to 0, no state is sent back to the client when adding a listener, nor it gets state when the node where the listener is registered changes.
Key/value filter factory name	string	Optional name of the key/value filter factory to be used with this listener. The factory is used to create key/value filter instances which allow events to be filtered directly in the Hot Rod server, avoiding sending events that the client is not interested in. If no factory is to be used, the length of the string is 0.
Key/value filter factory parameter count	byte	The key/value filter factory, when creating a filter instance, can take an arbitrary number of parameters, enabling the factory to be used to create different filter instances dynamically. This count field indicates how many parameters will be passed to the factory. If no factory name was provided, this field is not present in the request.
Key/value filter factory parameter 1	byte array	First key/value filter factory parameter
Key/value filter factory parameter 2	byte array	Second key/value filter factory parameter
…		
Converter factory name	string	Optional name of the converter factory to be used with this listener. The factory is used to transform the contents of the events sent to clients. By default, when no converter is in use, events are well defined, according to the type of event generated. However, there might be situations where users want to add extra information to the event, or they want to reduce the size of the events. In these cases, a converter can be used to transform the event contents. The given converter factory name produces converter instances to do this job. If no factory is to be used, the length of the string is 0.
Converter factory parameter count	byte	The converter factory, when creating a converter instance, can take an arbitrary number of parameters, enabling the factory to be used to create different converter instances dynamically. This count field indicates how many parameters will be passed to the factory. If no factory name was provided, this field is not present in the request.
Converter factory parameter 1	byte array	First converter factory parameter

Field Name	Size	Value
Converter factory parameter 2	byte array	Second converter factory parameter
…		

Response (0x26):

Field Name	Size	Value
Header	variable	Response header

Remove client listener for remote events

Request (0x27):

Field Name	Size	Value
Header	variable	Request header
Listener ID	byte array	Listener identifier

Response (0x28):

Field Name	Size	Value
Header	variable	Response header

Size

Request (0x29):

Field Name	Size	Value
Header	variable	Request header

Response (0x2A):

Field Name	Size	Value
Header	variable	Response header
Size	vInt	Size of the remote cache, which is calculated globally in the clustered set ups, and if present, takes cache store contents into account as well.

Exec

Request (0x2B):

Field Name	Size	Value
Header	variable	Request header
Script	string	Name of the task to execute

Field Name	Size	Value
Parameter Count	vInt	The number of parameters
Parameter 1 Name	string	The name of the first parameter
Parameter 1 Length	vInt	The length of the first parameter
Parameter 1 Value	byte array	The value of the first parameter

Response (0x2C):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if execution completed successfully 0x85 = server error
Value Length	vInt	If success, length of return value
Value	byte array	If success, the result of the execution

1.5.5. Remote Events

Starting with Hot Rod 2.0, clients can register listeners for remote events happening in the server. Sending these events commences the moment a client adds a client listener for remote events.

Event Header:

Field Name	Size	Value
Magic	1 byte	0xA1 = response
Message ID	vLong	ID of event
Opcode	1 byte	Event type: 0x60 = cache entry created event 0x61 = cache entry modified event 0x62 = cache entry removed event 0x66 = counter event 0x50 = error
Status	1 byte	Status of the response, possible values: 0x00 = No error
Topology Change Marker	1 byte	Since events are not associated with a particular incoming topology ID to be able to decide whether a new topology is required to be sent or not, new topologies will never be sent with events. Hence, this marker will always have `0` value for events.

Table 3. Cache entry created event

Field Name	Size	Value
Header	variable	Event header with `0x60` operation code
Listener ID	byte array	Listener for which this event is directed
Custom marker	byte	Custom event marker. For created events, this is `0`.
Command retried	byte	Marker for events that are result of retried commands. If command is retried, it returns `1`, otherwise `0`.
Key	byte array	Created key
Version	long	Version of the created entry. This version information can be used to make conditional operations on this cache entry.

Table 4. Cache entry modified event

Field Name	Size	Value
Header	variable	Event header with `0x61` operation code
Listener ID	byte array	Listener for which this event is directed
Custom marker	byte	Custom event marker. For created events, this is `0`.
Command retried	byte	Marker for events that are result of retried commands. If command is retried, it returns `1`, otherwise `0`.
Key	byte array	Modified key
Version	long	Version of the modified entry. This version information can be used to make conditional operations on this cache entry.

Table 5. Cache entry removed event

Field Name	Size	Value
Header	variable	Event header with `0x62` operation code
Listener ID	byte array	Listener for which this event is directed
Custom marker	byte	Custom event marker. For created events, this is `0`.
Command retried	byte	Marker for events that are result of retried commands. If command is retried, it returns `1`, otherwise `0`.
Key	byte array	Removed key

Table 6. Custom event

Field Name	Size	Value
Header	variable	Event header with event specific operation code
Listener ID	byte array	Listener for which this event is directed
Custom marker	byte	Custom event marker. For custom events, this is `1`.
Event data	byte array	Custom event data, formatted according to the converter implementation logic.

1.6. Hot Rod Protocol 2.1

Infinispan versions

This version of the protocol is implemented since Infinispan 7.1.0.Final.

1.6.1. Request Header

The `version` field in the header is updated to `21`.

1.6.2. Operations

Add client listener for remote events

An extra byte parameter is added at the end which indicates whether the client prefers client listener to work with raw binary data for filter/converter callbacks. If using raw data, its value is `1` otherwise `0`.

Request format:

Field Name	Size	Value
Header	variable	Request header
Listener ID	byte array	…
Include state	byte	…
Key/value filter factory parameter count	byte	…
…		
Converter factory name	string	…
Converter factory parameter count	byte	…
…		
Use raw data	byte	If filter/converter parameters should be raw binary, then `1`, otherwise `0`.

Custom event

Starting with Hot Rod 2.1, custom events can return raw data that the Hot Rod client should not try to unmarshall before passing it on to the user. The way this is transmitted to the Hot Rod client is by sending `2` as the custom event marker. So, the format of the custom event remains like this:

Field Name	Size	Value
Header	variable	Event header with event specific operation code
Listener ID	byte array	Listener for which this event is directed

Field Name	Size	Value
Custom marker	byte	Custom event marker. For custom events whose event data needs to be unmarshalled before returning to user the value is 1. For custom events that need to return the event data as-is to the user, the value is 2.
Event data	byte array	Custom event data. If the custom marker is 1, the bytes represent the marshalled version of the instance returned by the converter. If custom marker is 2, it represents the byte array, as returned by the converter.

1.7. Hot Rod Protocol 2.2

Infinispan versions

This version of the protocol is implemented since Infinispan 8.0

Added support for different time units.

1.7.1. Operations

Put/PutAll/PutIfAbsent/Replace/ReplaceIfUnmodified

Common request format:

Field Name	Size	Value
TimeUnits	Byte	Time units of lifespan (first 4 bits) and maxIdle (last 4 bits). Special units DEFAULT and INFINITE can be used for default server expiration and no expiration respectively. Possible values: 0x00 = SECONDS 0x01 = MILLISECONDS 0x02 = NANOSECONDS 0x03 = MICROSECONDS 0x04 = MINUTES 0x05 = HOURS 0x06 = DAYS 0x07 = DEFAULT 0x08 = INFINITE
Lifespan	vLong	Duration which the entry is allowed to life. Only sent when time unit is not DEFAULT or INFINITE
Max Idle	vLong	Duration that each entry can be idle before it's evicted from the cache. Only sent when time unit is not DEFAULT or INFINITE

1.8. Hot Rod Protocol 2.3

Infinispan versions

This version of the protocol is implemented since Infinispan 8.0

1.8.1. Operations

Iteration Start

Request (0x31):

Field Name	Size	Value
Segments size	signed vInt	Size of the bitset encoding of the segments ids to iterate on. The size is the maximum segment id rounded to nearest multiple of 8. A value -1 indicates no segment filtering is to be done
Segments	byte array	(Optional) Contains the segments ids bitset encoded, where each bit with value 1 represents a segment in the set. Byte order is little-endian. Example: segments [1,3,12,13] would result in the following encoding: 00001010 00110000 size: 16 bits first byte: represents segments from 0 to 7, from which 1 and 3 are set second byte: represents segments from 8 to 15, from which 12 and 13 are set More details in the java.util.BitSet implementation. Segments will be sent if the previous field is not negative
FilterConverter size	signed vInt	The size of the String representing a KeyValueFilterConverter factory name deployed on the server, or -1 if no filter will be used
FilterConverter	UTF-8 byte array	(Optional) KeyValueFilterConverter factory name deployed on the server. Present if previous field is not negative
BatchSize	vInt	number of entries to transfers from the server at one go

Response (0x32):

Field Name	Size	Value
IterationId	String	The unique id of the iteration

Iteration Next

Request (0x33):

Field Name	Size	Value
IterationId	String	The unique id of the iteration

Response (0x34):

Field Name	Size	Value
Finished segments size	vInt	size of the bitset representing segments that were finished iterating
Finished segments	byte array	bitset encoding of the segments that were finished iterating
Entry count	vInt	How many entries are being returned
Key 1 Length	vInt	Length of key
Key 1	byte array	Retrieved key
Value 1 Length	vInt	Length of value
Value 1	byte array	Retrieved value
Key 2 Length	vInt	
Key 2	byte array	
Value 2 Length	vInt	
Value 2	byte array	
… continues until entry count is reached		

Iteration End

Request (0x35):

Field Name	Size	Value
IterationId	String	The unique id of the iteration

Response (0x36):

Header	variable	Response header
Response status	1 byte	0x00 = success, if execution completed successfully 0x05 = for non existent IterationId

1.9. Hot Rod Protocol 2.4

Infinispan versions

This version of the protocol is implemented since Infinispan 8.1

This Hot Rod protocol version adds three new status code that gives the client hints on whether the server has compatibility mode enabled or not:

- `0x06`: Success status and compatibility mode is enabled.
- `0x07`: Success status and return previous value, with compatibility mode is enabled.
- `0x08`: Not executed and return previous value, with compatibility mode is enabled.

The Iteration Start operation can optionally send parameters if a custom filter is provided and it's parametrised:

1.9.1. Operations

Iteration Start

Request (0x31):

Field Name	Size	Value
Segments size	signed vInt	same as protocol version 2.3.
Segments	byte array	same as protocol version 2.3.
FilterConverter size	signed vInt	same as protocol version 2.3.
FilterConverter	UTF-8 byte array	same as protocol version 2.3.
Parameters size	byte	the number of params of the filter. Only present when FilterConverter is provided.
Parameters	byte[][]	an array of parameters, each parameter is a byte array. Only present if Parameters size is greater than 0.
BatchSize	vInt	same as protocol version 2.3.

The Iteration Next operation can optionally return projections in the value, meaning more than one value is contained in the same entry.

Iteration Next

Response (0x34):

Field Name	Size	Value
Finished segments size	vInt	same as protocol version 2.3.
Finished segments	byte array	same as protocol version 2.3.
Entry count	vInt	same as protocol version 2.3.
Number of value projections	vInt	Number of projections for the values. If 1, behaves like version protocol version 2.3.
Key1 Length	vInt	same as protocol version 2.3.
Key1	byte array	same as protocol version 2.3.
Value1 projection1 length	vInt	length of value1 first projection
Value1 projection1	byte array	retrieved value1 first projection
Value1 projection2 length	vInt	length of value2 second projection

Field Name	Size	Value
Value1 projection2	byte array	retrieved value2 second projection
… continues until all projections for the value retrieved	Key2 Length	vInt
same as protocol version 2.3.	Key2	byte array
same as protocol version 2.3.	Value2 projection1 length	vInt
length of value 2 first projection	Value2 projection1	byte array
retrieved value 2 first projection	Value2 projection2 length	vInt
length of value 2 second projection	Value2 projection2	byte array
retrieved value 2 second projection	… continues until entry count is reached	

1. Stats:

Statistics returned by previous Hot Rod protocol versions were local to the node where the Hot Rod operation had been called. Starting with 2.4, new statistics have been added which provide global counts for the statistics returned previously. If the Hot Rod is running in local mode, these statistics are not returned:

Name	Explanation
globalCurrentNumberOfEntries	Number of entries currently across the Hot Rod cluster.
globalStores	Total number of put operations across the Hot Rod cluster.
globalRetrievals	Total number of get operations across the Hot Rod cluster.
globalHits	Total number of get hits across the Hot Rod cluster.
globalMisses	Total number of get misses across the Hot Rod cluster.

Name	Explanation
globalRemoveHits	Total number of removal hits across the Hot Rod cluster.
globalRemoveMisses	Total number of removal misses across the Hot Rod cluster.

1.10. Hot Rod Protocol 2.5

Infinispan versions

This version of the protocol is implemented since Infinispan 8.2

This Hot Rod protocol version adds support for metadata retrieval along with entries in the iterator. It includes two changes:

- Iteration Start request includes an optional flag
- IterationNext operation may include metadata info for each entry if the flag above is set

Iteration Start

Request (0x31):

Field Name	Size	Value
Segments size	signed vInt	same as protocol version 2.4.
Segments	byte array	same as protocol version 2.4.
FilterConverter size	signed vInt	same as protocol version 2.4.
FilterConverter	UTF-8 byte array	same as protocol version 2.4.
Parameters size	byte	same as protocol version 2.4.
Parameters	byte[][]	same as protocol version 2.4.
BatchSize	vInt	same as protocol version 2.4.
Metadata	1 byte	1 if metadata is to be returned for each entry, 0 otherwise

Iteration Next

Response (0x34):

Field Name	Size	Value
Finished segments size	vInt	same as protocol version 2.4.
Finished segments	byte array	same as protocol version 2.4.
Entry count	vInt	same as protocol version 2.4.

Field Name	Size	Value
Number of value projections	vInt	same as protocol version 2.4.
Metadata (entry 1)	1 byte	If set, entry has metadata associated
Expiration (entry 1)	1 byte	A flag indicating whether the response contains expiration information. The value of the flag is obtained as a bitwise OR operation between INFINITE_LIFESPAN (0x01) and `INFINITE_MAXIDLE (0x02)`. Only present if the metadata flag above is set
Created (entry 1)	Long	(optional) a Long representing the timestamp when the entry was created on the server. This value is returned only if the flag's INFINITE_LIFESPAN bit is not set.
Lifespan (entry 1)	vInt	(optional) a vInt representing the lifespan of the entry in seconds. This value is returned only if the flag's INFINITE_LIFESPAN bit is not set.
LastUsed (entry 1)	Long	(optional) a Long representing the timestamp when the entry was last accessed on the server. This value is returned only if the flag's `INFINITE_MAXIDLE` bit is not set.
MaxIdle (entry 1)	vInt	(optional) a vInt representing the maxIdle of the entry in seconds. This value is returned only if the flag's `INFINITE_MAXIDLE` bit is not set.
Entry Version (entry 1)	8 bytes	Unique value of an existing entry's modification. Only present if Metadata flag is set
Key 1 Length	vInt	same as protocol version 2.4.
Key 1	byte array	same as protocol version 2.4.
Value 1 Length	vInt	same as protocol version 2.4.
Value 1	byte array	same as protocol version 2.4.
Metadata (entry 2)	1 byte	Same as for entry 1
Expiration (entry 2)	1 byte	Same as for entry 1
Created (entry 2)	Long	Same as for entry 1
Lifespan (entry 2)	vInt	Same as for entry 1
LastUsed (entry 2)	Long	Same as for entry 1
MaxIdle (entry 2)	vInt	Same as for entry 1
Entry Version (entry 2)	8 bytes	Same as for entry 1
Key 2 Length	vInt	
Key 2	byte array	
Value 2 Length	vInt	

Field Name	Size	Value
Value 2	byte array	
… continues until entry count is reached		

1.11. Hot Rod Protocol 2.6

Infinispan versions

This version of the protocol is implemented since Infinispan 9.0

This Hot Rod protocol version adds support for streaming get and put operations. It includes two new operations:

- GetStream for retrieving data as a stream, with an optional initial offset
- PutStream for writing data as a stream, optionally by specifying a version

GetStream

Request (0x37):

Field Name	Size	Value
Header	variable	Request header
Offset	vInt	The offset in bytes from which to start retrieving. Set to 0 to retrieve from the beginning
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.

GetStream

Response (0x38):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if key retrieved 0x02 = if key does not exist
Flag	1 byte	A flag indicating whether the response contains expiration information. The value of the flag is obtained as a bitwise OR operation between INFINITE_LIFESPAN (0x01) and `INFINITE_MAXIDLE (0x02)`.

Field Name	Size	Value
Created	Long	(optional) a Long representing the timestamp when the entry was created on the server. This value is returned only if the flag's INFINITE_LIFESPAN bit is not set.
Lifespan	vInt	(optional) a vInt representing the lifespan of the entry in seconds. This value is returned only if the flag's INFINITE_LIFESPAN bit is not set.
LastUsed	Long	(optional) a Long representing the timestamp when the entry was last accessed on the server. This value is returned only if the flag's `INFINITE_MAXIDLE` bit is not set.
MaxIdle	vInt	(optional) a vInt representing the maxIdle of the entry in seconds. This value is returned only if the flag's `INFINITE_MAXIDLE` bit is not set.
Entry Version	8 bytes	Unique value of an existing entry's modification. The protocol does not mandate that entry_version values are sequential. They just need to be unique per update at the key level.
Value Length	vInt	If success, length of value
Value	byte array	If success, the requested value

PutStream

Request (0x39)

Field Name	Size	Value
Header	variable	Request header
Entry Version	8 bytes	Possible values 0 = Unconditional put -1 = Put If Absent Other values = pass a version obtained by `GetWithMetadata` operation to perform a conditional replace.
Key Length	vInt	Length of key. Note that the size of a vint can be up to 5 bytes which in theory can produce bigger numbers than Integer.MAX_VALUE. However, Java cannot create a single array that's bigger than Integer.MAX_VALUE, hence the protocol is limiting vint array lengths to Integer.MAX_VALUE.
Key	byte array	Byte array containing the key whose value is being requested.
Value Chunk 1 Length	vInt	The size of the first chunk of data. If this value is 0 it means the client has completed transferring the value and the operation should be performed.
Value Chunk 1	byte array	Array of bytes forming the fist chunk of data.

Field Name	Size	Value
…continues until the value is complete		

Response (0x3A):

Field Name	Size	Value
Header	variable	Response header

On top of these additions, this Hot Rod protocol version improves remote listener registration by adding a byte that indicates at a global level, which type of events the client is interested in. For example, a client can indicate that only created events, or only expiration and removal events…etc. More fine grained event interests, e.g. per key, can be defined using the key/value filter parameter.

So, the new add listener request looks like this:

Add client listener for remote events

Request (0x25):

Field Name	Size	Value
Header	variable	Request header
Listener ID	byte array	Listener identifier
Include state	byte	When this byte is set to `1`, cached state is sent back to remote clients when either adding a cache listener for the first time, or when the node where a remote listener is registered changes in a clustered environment. When enabled, state is sent back as cache entry created events to the clients. If set to `0`, no state is sent back to the client when adding a listener, nor it gets state when the node where the listener is registered changes.
Key/value filter factory name	string	Optional name of the key/value filter factory to be used with this listener. The factory is used to create key/value filter instances which allow events to be filtered directly in the Hot Rod server, avoiding sending events that the client is not interested in. If no factory is to be used, the length of the string is `0`.
Key/value filter factory parameter count	byte	The key/value filter factory, when creating a filter instance, can take an arbitrary number of parameters, enabling the factory to be used to create different filter instances dynamically. This count field indicates how many parameters will be passed to the factory. If no factory name was provided, this field is not present in the request.
Key/value filter factory parameter 1	byte array	First key/value filter factory parameter

Field Name	Size	Value
Key/value filter factory parameter 2	byte array	Second key/value filter factory parameter
…		
Converter factory name	string	Optional name of the converter factory to be used with this listener. The factory is used to transform the contents of the events sent to clients. By default, when no converter is in use, events are well defined, according to the type of event generated. However, there might be situations where users want to add extra information to the event, or they want to reduce the size of the events. In these cases, a converter can be used to transform the event contents. The given converter factory name produces converter instances to do this job. If no factory is to be used, the length of the string is `0`.
Converter factory parameter count	byte	The converter factory, when creating a converter instance, can take an arbitrary number of parameters, enabling the factory to be used to create different converter instances dynamically. This count field indicates how many parameters will be passed to the factory. If no factory name was provided, this field is not present in the request.
Converter factory parameter 1	byte array	First converter factory parameter
Converter factory parameter 2	byte array	Second converter factory parameter
…		
Listener even type interests	vInt	A variable length number representing listener event type interests. Each event type is represented by a bit. Each flags is represented by a bit. Note that since this field is sent as variable length, the most significant bit in a byte is used to determine whether more bytes need to be read, hence this bit does not represent any flag. Using this model allows for flags to be combined in a short space. Here are the current values for each flag: 0x01 = cache entry created events 0x02 = cache entry modified events 0x04 = cache entry removed events 0x08 = cache entry expired events

1.12. Hot Rod Protocol 2.7

Infinispan versions

This version of the protocol is implemented since Infinispan 9.2

This Hot Rod protocol version adds support for transaction operations. It includes 3 new operations:

- Prepare, with the transaction write set (i.e. modified keys), it tries to prepare and validate the transaction in the server.
- Commit, commits a prepared transaction.
- Rollback, rollbacks a prepared transaction.

Prepare Request

Request (0x3B):

Field Name	Size	Value
Header	variable	Request header
Xid	XID	The transaction ID (XID)
OnePhaseCommit	byte	When it is set to 1, the server will use one-phase-commit if available (XA only)
Number of keys	vInt	The number of keys
For each key (keys must be distinct)		
Key Length	vInt	Length of key. Note that the size of a vInt can be up to 5 bytes which in theory can produce bigger numbers than `Integer.MAX_VALUE`. However, Java cannot create a single array that's bigger than `Integer.MAX_VALUE`, hence the protocol is limiting vInt array lengths to `Integer.MAX_VALUE`.
Key	byte array	Byte array containing the key
Control Byte	Byte	A bit set with the following meaning: 0x01 = `NOT_READ` 0x02 = `NON_EXISTING` 0x04 = `REMOVE_OPERATION` Note that `NOT_READ` and `NON_EXISTING` can't be set at the same time.
Version Read	long	The version read. Only sent when `NOT_READ` and `NON_EXISTING` aren't present.

Field Name	Size	Value
TimeUnits	Byte	Time units of lifespan (first 4 bits) and maxIdle (last 4 bits). Special units `DEFAULT` and `INFINITE` can be used for default server expiration and no expiration respectively. Possible values: 0x00 = `SECONDS` 0x01 = `MILLISECONDS` 0x02 = `NANOSECONDS` 0x03 = `MICROSECONDS` 0x04 = `MINUTES` 0x05 = `HOURS` 0x06 = `DAYS` 0x07 = `DEFAULT` 0x08 = `INFINITE` Only sent when `REMOVE_OPERATION` isn't set.
Lifespan	vLong	Duration which the entry is allowed to life. Only sent when time unit is not `DEFAULT` or `INFINITE` and `REMOVE_OPERATION` isn't set.
Max Idle	vLong	Duration that each entry can be idle before it's evicted from the cache. Only sent when time unit is not `DEFAULT` or `INFINITE` and `REMOVE_OPERATION` isn't set.
Value Length	vInt	Length of value. Only sent if `REMOVE_OPERATION` isn't set.
Value	byte-array	Value to be stored. Only sent if `REMOVE_OPERATION` isn't set.

Commit and Rollback Request

Request. Commit (0x3D) and Rollback (0x3F):

Field Name	Size	Value
Header	variable	Request header
Xid	XID	The transaction ID (XID)

Response from prepare, commit and rollback request.

Response. Prepare (0x3C), Commit (0x3E) and Rollback (0x40)

Field Name	Size	Value
Header	variable	Response header
XA return code	vInt	The XA code representing the prepare response. Can be `XA_OK(0)`, `XA_RDONLY(3)` or any of the error codes (see `XaException`). This field isn't present if the response state is different from `Successful`.

XID Format

The XID in the requests has the following format:

Field Name	Size	Value
Format ID	signed vInt	The XID format.
Length of Global Transaction id	byte	The length of global transaction id byte array. It max value is `64`.
Global Transaction Id	byte array	The global transaction id.
Length of Branch Qualifier	byte	The length of branch qualifier byte array. It max value is `64`.
Branch Qualifier	byte array	The branch qualifier.

Counter Configuration encoding format

The `CounterConfiguration` class encoding format is the following:

In counter related operation, the `Cache Name` field in Request Header can be empty.

Summary of `Status` value in the Response Header:
* `0x00`: Operation successful.
* `0x01`: Operation failed.
* `0x02`: The counter isn't defined.
* `0x04`: The counter reached a boundary. Only possible for `STRONG` counters.

Field Name	Size	Value
Flags	byte	The `CounterType` and `Storage` encoded. Only the less significant bits are used as following: 1st bit: `1` for `WEAK` counter and `0` for `STRONG` counter. 2nd bit: `1` for `BOUNDED` counter and `0` for `UNBOUNDED` counter 3rd bit: `1` for `PERSISTENT` storage and `0` for `VOLATILE` storage.
Concurrency Level	vInt	(Optional) the counter's concurrency-level. Only present if the counter is `WEAK`.
Lower bound	long	(Optional) the lower bound of a bounded counter. Only present if the counter is `BOUNDED`.
Upper bound	long	(Optional) the upper bound of a bounded counter. Only present if the counter is `BOUNDED`.
Initial value	long	The counter's initial value.

Counter create operation

Creates a counter if it doesn't exist.

Table 7. Request (0x4B)

Field Name	Size	Value
Header	variable	Request header

Field Name	Size	Value
Name	string	The counter's name
Counter Configuration	variable	The counter's configuration. See CounterConfiguration encode.

Table 8. Response (0x4C)

Field Name	Size	Value
Header	variable	Response header

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x01`: Operation failed. Counter is already defined.
- See the Reponse Header for error codes.

Counter get configuration operation

Returns the counter's configuration.

Table 9. Request (0x4D)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name.

Table 10. Response (0x4E)

Field Name	Size	Value
Header	variable	Response header
Counter Configuration	variable	(Optional) The counter's configuration. Only present if `Status==0x00`. See CounterConfiguration encode.

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x02`: Counter doesn't exist.
- See the Reponse Header for error codes.

Counter is defined operation

Checks if the counter is defined.

Table 11. Request (0x4F)

Field Name	Size	Value
Header	variable	Request header

Field Name	Size	Value
Name	string	The counter's name

Table 12. Response (0x51)

Field Name	Size	Value
Header	variable	Response header

Response Header `Status` possible values:

- `0x00`: Counter is defined.
- `0x01`: Counter isn't defined.
- See the Reponse Header for error codes.

Counter add-and-get operation

Adds a value to the counter and returns the new value.

Table 13. Request (0x52)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name
Value	long	The value to add

Table 14. Response (0x53)

Field Name	Size	Value
Header	variable	Response header
Value	long	(Optional) the counter's new value. Only present if `Status==0x00`.

Since the `WeakCounter` doesn't have access to the new value, the `value` is zero.

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x02`: The counter isn't defined.
- `0x04`: The counter reached its boundary. Only possible for `STRONG` counters.
- See the Reponse Header for error codes.

Counter reset operation

Resets the counter's value.

Table 15. Request (0x54)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name

Table 16. Response (0x55)

Field Name	Size	Value
Header	variable	Response header

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x02`: Counter isn't defined.
- See the Reponse Header for error codes.

Counter get operation

Returns the counter's value.

Table 17. Request (0x56)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name

Table 18. Response (0x57)

Field Name	Size	Value
Header	variable	Response header
Value	long	(Optional) the counter's value. Only present if `Status==0x00`.

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x02`: Counter isn't defined.
- See the Reponse Header for error codes.

Counter compare-and-swap operation

Compares and only updates the counter value if the current value is the expected.

Table 19. Request (0x58)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name
Expect	long	The counter's expected value.

Field Name	Size	Value
Update	long	The counter's value to set.

Table 20. Response (0x59)

Field Name	Size	Value
Header	variable	Response header
Value	long	(Optional) the counter's value. Only present if `Status==0x00`.

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x02`: The counter isn't defined.
- `0x04`: The counter reached its boundary. Only possible for `STRONG` counters.
- See the Reponse Header for error codes.

Counter add and remove listener

Adds/Removes a listener for a counter

Table 21. Request ADD (0x5A) / REMOVE (0x5C)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name
Listener-id	byte array	The listener's id

Table 22. Response: ADD (0x5B) / REMOVE (0x5D)

Field Name	Size	Value
Header	variable	Response header

Response Header `Status` possible values:

- `0x00`: Operation successful and the connection used in the request will be used to send event (add) or the connection can be removed (remove).
- `0x01`: Operation successful and the current connection is still in use.
- `0x02`: The counter isn't defined.
- See the Reponse Header for error codes.

Table 23. Counter Event (0x66)

Field Name	Size	Value
Header	variable	Event header with operation code `0x66`
Name	string	The counter's name

Field Name	Size	Value
Listener-id	byte array	The listener's id
Encoded Counter State	byte	Encoded old and new counter state. Bit set: `------00`: Valid old state `------01`: Lower bound reached old state `------10`: Upper bound reached old state `----00--`: Valid new state `----01--`: Lower bound reached new state `----10--`: Upper bound reached new state
Old value	long	Counter's old value
New value	long	Counter's new value

All counters under a `CounterManager` implementation can use the same `listener-id`.

A connection is dedicated to a single `listener-id` and can receive events from different counters.

Counter remove operation

Removes the counter from the cluster.

The counter is re-created if it is accessed again.

Table 24. Request (0x5E)

Field Name	Size	Value
Header	variable	Request header
Name	string	The counter's name

Table 25. Response (0x5F)

Field Name	Size	Value
Header	variable	Response header

Response Header `Status` possible values:

- `0x00`: Operation successful.
- `0x02`: The counter isn't defined.
- See the Reponse Header for error codes.

1.13. Hot Rod Protocol 2.8

Infinispan versions

This version of the protocol is implemented since Infinispan 9.3

Events

The protocol allows clients to send requests on the same connection that was previously used for Add Client Listener operation, and in protocol < 2.8 is reserved for sending events to the client. This includes registering additional listeners, therefore receiving events for multiple listeners.

The binary format of requests/responses/events does not change but the previously meaningless `messageId` in events must be set to:

- `messageId` of the Add Client Listener operation for the include-current-state events
- `0` for the events sent after the Add Client Listener operation has been finished (response sent).

The same holds for counter events: client can send further requests after Counter Add Listener. Previously meaningless `messageId` in counter event is always set to `0`.

These modifications of the protocol do not require any changes on the client side (as the client simply won't send additional operations if it does not support that; the changes are more permissive to the clients) but the server has to handle load on the connection correctly.

MediaType

This Hot Rod protocol version also adds support for specifying the MediaType of Keys and Values, allowing data to be read (and written) in different formats. This information is part of the Header.

The data formats are described using a *MediaType* object, that is represented as follows:

Field Name	Size	Value
type	1 byte	0x00 = No MediaType supplied 0x01 = Pre-defined MediaType supplied 0x02 = Custom MediaType supplied
id	vInt	(Optional) For a pre-defined MediaType (type=0x01), the Id of the MediaType. The currently supported Ids can be found at MediaTypeIds
customString	string	(Optional) If a custom MediaType is supplied (type=0x02), the custom MediaType of the key, including type and subtype. E.g.: *text/plain*, *application/json*, etc.
paramSize	vInt	The size of the parameters for the MediaType
paramKey1	string	(Optional) The first parameter's key
paramValue1	string	(Optional) The first parameter's value
…	…	…
paramKeyN	string	(Optional) The nth parameter's key
paramValueN	string	(Optional) The nth parameter's value

1.13.1. Request Header

The request header has the following extra fields:

Field Name	Type	Value
Key Format	MediaType	The MediaType to be used for keys during the operation. It applies to both the keys sent and received.
Value Format	MediaType	Analogous to Key Format, but applied for the values.

1.14. Hot Rod Protocol 2.9

Infinispan versions

This version of the protocol is implemented since Infinispan 9.4

Compatibility Mode removal

The compatibility mode hint from the *Response status* fields from the operations is not sent anymore. Consequently, the following statuses are removed:

- `0x06`: Success status with compatibility mode.
- `0x07`: Success status with return previous value and compatibility mode.
- `0x08`: Not executed with return previous value and compatibility mode.

To figure out what is the server's storage, the configured MediaType of keys and values are returned on the ping operation:

Ping Response (0x18):

Field Name	Size	Value
Header	variable	same as before
Response status	1 byte	same as before
Key Type	MediaType	Media Type of the key stored in the server
Value Type	MediaType	Media Type of the value stored in the server

New query format

This version supports query requests and responses in JSON format. The format of the operations **0x1F** (Query Request) and **0x20** (Query Response) are not changed.

To send JSON payloads, the "Value Format" field in the header should be *application/json*.

Query Request (0x1F):

Field Name	Size	Value
Header	variable	Request header
Query Length	vInt	The length of the UTF-8 encoded query object.

Field Name	Size	Value

Query	byte array	Byte array containing the JSON (UTF-8) encoded query object, having a length specified by the previous field. Example of payload:
		
		`{ "query":"From Entity where field1:'value1'", "offset": 12, "max-results": 1000, "query-mode": "FETCH" }`
		
		Where:
		
		`query: the Ickle query String. offset: the index of the first result to return. max_results: the maximum number of results to return. query_mode: the indexed query mode. Either FETCH or BROADCAST. FECTH is the default.`

Query Response (0x20):

Field Name	Size	Value
Header	variable	Response header
Response payload Length	vInt	The length of the UTF-8 encoded response object

Field Name	Size	Value

Response payload	byte array	Byte array containing the JSON encoded response object, having a length specified by previous field. Example payload:
		
		`{ "total_results":801, "hits":[ { "hit":{ "field1":565, "field2":"value2" } }, { "hit":{ "field1":34, "field2":"value22" } } ] }`
		
		Where:
		
		`total_results: the total number of results of the query. hits: an ARRAY of OBJECT representing the results. hit: each OBJECT above contain another OBJECT in the "hit" field, containing the result of the query, in JSON format.`

Also, this version introduces 3 new operations for Hot Rod transactions:

- Prepare Request V2: It adds new parameters to the request. The response stays the same.
- Forget Transaction Request: Removes transaction information in the server.
- Fetch In-Doubt Transactions Request: Fetches all in-doubt transactions's Xid.

Prepare Request V2

Request (0x7D):

Field Name	Size	Value
Header	variable	Request header
Xid	XID	The transaction ID (XID)

Field Name	Size	Value
OnePhaseCommit	byte	When it is set to 1, the server will use one-phase-commit if available (XA only)
Recoverable	byte	Set to 1 to allow recovery in this transactions
Timeout	long	The idle timeout in milliseconds. If the transaction isn't recoverable (`Recoverable=0`), the server rollbacks the transaction if it has been idle for this amount of time.
Number of keys	vInt	The number of keys
For each key (keys must be distinct)		
Key Length	vInt	Length of key. Note that the size of a vInt can be up to 5 bytes which in theory can produce bigger numbers than `Integer.MAX_VALUE`. However, Java cannot create a single array that's bigger than `Integer.MAX_VALUE`, hence the protocol is limiting vInt array lengths to `Integer.MAX_VALUE`.
Key	byte array	Byte array containing the key
Control Byte	Byte	A bit set with the following meaning: 0x01 = `NOT_READ` 0x02 = `NON_EXISTING` 0x04 = `REMOVE_OPERATION` Note that `NOT_READ` and `NON_EXISTING` can't be set at the same time.
Version Read	long	The version read. Only sent when `NOT_READ` and `NON_EXISTING` aren't present.
TimeUnits	Byte	Time units of lifespan (first 4 bits) and maxIdle (last 4 bits). Special units `DEFAULT` and `INFINITE` can be used for default server expiration and no expiration respectively. Possible values: 0x00 = `SECONDS` 0x01 = `MILLISECONDS` 0x02 = `NANOSECONDS` 0x03 = `MICROSECONDS` 0x04 = `MINUTES` 0x05 = `HOURS` 0x06 = `DAYS` 0x07 = `DEFAULT` 0x08 = `INFINITE` Only sent when `REMOVE_OPERATION` isn't set.
Lifespan	vLong	Duration which the entry is allowed to life. Only sent when time unit is not `DEFAULT` or `INFINITE` and `REMOVE_OPERATION` isn't set.
Max Idle	vLong	Duration that each entry can be idle before it's evicted from the cache. Only sent when time unit is not `DEFAULT` or `INFINITE` and `REMOVE_OPERATION` isn't set.
Value Length	vInt	Length of value. Only sent if `REMOVE_OPERATION` isn't set.
Value	byte-array	Value to be stored. Only sent if `REMOVE_OPERATION` isn't set.

Response (0x7E)

Field Name	Size	Value
Header	variable	Response header
XA return code	vInt	The XA code representing the prepare response. Can be `XA_OK(0)`, `XA_RDONLY(3)` or any of the error codes (see `XaException`). This field isn't present if the response state is different from `Successful`.

Forget Transaction

Request (0x79)

Field Name	Size	Value
Header	variable	Request header
Xid	XID	The transaction ID (XID)

Response (0x7A)

Field Name	Size	Value
Header	variable	Response header

Fetch in-doubt transactions

Request (0x7B)

Field Name	Size	Value
Header	variable	Request header

Response (0x7C)

Field Name	Size	Value
Header	variable	Response header
Number of Xid	vInt	The number of Xid in response
For each entry:		
Xid	XID	The transaction ID (XID)

1.15. Hot Rod Hash Functions

Infinispan makes use of a consistent hash function to place nodes on a hash wheel, and to place keys of entries on the same wheel to determine where entries live.

In Infinispan 4.2 and earlier, the hash space was hardcoded to 10240, but since 5.0, the hash space is Integer.MAX_INT . Please note that since Hot Rod clients should not assume a particular hash

space by default, every time a hash-topology change is detected, this value is sent back to the client via the Hot Rod protocol.

When interacting with Infinispan via the Hot Rod protocol, it is mandated that keys (and values) are byte arrays, to ensure platform neutral behavior. As such, smart-clients which are aware of hash distribution on the backend would need to be able to calculate the hash codes of such byte array keys, again in a platform-neutral manner. To this end, the hash functions used by Infinispan are versioned and documented, so that it can be re-implemented by non-Java clients if needed.

The version of the hash function in use is provided in the Hot Rod protocol, as the hash function version parameter.

1. Version 1 (single byte, 0x01) The initial version of the hash function in use is based on Austin Appleby's MurmurHash 2.0 algorithm , a fast, non-cryptographic hash that exhibits excellent distribution, collision resistance and avalanche behavior. The specific version of the algorithm used is the slightly slower, endian-neutral version that allows consistent behavior across both big- and little-endian CPU architectures. Infinispan's version also hard-codes the hash seed as -1. For details of the algorithm, please visit Austin Appleby's MurmurHash 2.0 page. Other implementations are detailed on Wikipedia . This hash function was the default one used by the Hot Rod server until Infinispan 4.2.1. Since Infinispan 5.0, the server never uses hash version 1. Since Infinispan 9.0, the client ignores hash version 1.
2. Version 2 (single byte, 0x02) Since Infinispan 5.0, a new hash function is used by default which is based on Austin Appleby's MurmurHash 3.0 algorithm. Detailed information about the hash function can be found in this wiki. Compared to 2.0, it provides better performance and spread. Since Infinispan 7.0, the server only uses version 2 for HotRod 1.x clients.
3. Version 3 (single byte, 0x03) Since Infinispan 7.0, a new hash function is used by default. The function is still based on wiki, but is also aware of the hash segments used in the server's ConsistentHash.

1.16. Hot Rod Admin Tasks

Admin operations are handled by the Exec operation with a set of well known tasks. Admin tasks are named according to the following rules:

```
@@context@name
```

All parameters are UTF-8 encoded strings. Parameters are specific to each task, with the exception of the **flags** parameter which is common to all commands. The **flags** parameter contains zero or more space-separated values which may affect the behaviour of the command. The following table lists all currently available flags.

Admin tasks return the result of the operation represented as a JSON string.

Table 26. FLAGS

Flag	Description
permanent	Requests that the command's effect be made permanent into the server's configuration. If the server cannot comply with the request, the entire operation will fail with an error

1.16.1. Admin tasks

Table 27. @@cache@create

Parameter	Description	Required
name	The name of the cache to create.	Yes
template	The name of the cache configuration template to use for the new cache.	No
configuration	the XML declaration of a cache configuration to use.	No
flags	See the flags table above.	No

Table 28. @@cache@remove

Parameter	Description	Required
name	The name of the cache to remove.	Yes
flags	See the flags table above.	No

@@cache@names

Returns the cache names as a JSON array of strings, e.g. ["cache1", "cache2"]

Table 29. @@cache@reindex

Parameter	Description	Required
name	The name of the cache to reindex.	Yes
flags	See the flags table above.	No, all flags will be ignored

1.17. Hot Rod Protocol 3.0

Infinispan versions

This version of the protocol is implemented since Infinispan 10.0

Automatic protocol selection

This version introduces changes to the `ping` operation which returns some additional information when called with a protocol version of 30 or higher. This allows the server to tell the client the highest supported protocol version as well as a list of supported operations. After obtaining the response from the server, the client should use the highest possible version understood by the server for all subsequent requests.

Expiration

Expiration values larger than the number of milliseconds in 30 days are no longer treated as Unix time and are interpreted literally.

ping

Request (0x17):

Field Name	Size	Value
Header	variable	Request header

ping

Response (0x18):

Field Name	Size	Value
Header	variable	Response header
Response status	1 byte	0x00 = success, if no errors
Key Type	MediaType	Media Type of the key stored in the server
Value Type	MediaType	Media Type of the value stored in the server
Version	1 byte	Hot Rod server version.
opCount	vInt	Number of supported operations
opRequestCode1	1 byte	Request opcode of the first operation
…	…	…
opRequestCodeN	1 short	Request opcode of the nth operation

A new flag has been added:

0x0020 = used when an operation wants to skip notifications to the registered listeners