[MS-HMSHTTP]:

Hive Metastore HTTP Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
10/16/2019	1.0	New	Released new document.
7/30/2020	2.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction 6
1.1 Glossary 6
1.2 References 7
1.2.1 Normative References 7
1.2.2 Informative References 7
1.3 Overview 8
1.4 Relationship to Other Protocols 8
1.5 Prerequisites/Preconditions 8
1.6 Applicability Statement 9
1.7 Versioning and Capability Negotiation 9
1.8 Vendor-Extensible Fields 9
1.9 Standards Assignments 9

2 Messages 10
2.1 Transport 10
2.2 Common Data Types 10
2.2.1 Namespaces 10
2.2.2 HTTP Methods 10
2.2.3 URI Parameters 10
2.2.3.1 clusterIp 10
2.2.3.2 knoxPort 10
2.2.3.3 path 10
2.2.4 Data Structures 10
2.2.4.1 Database 11
2.2.4.2 Table 11
2.2.4.3 Partition 12
2.2.4.4 MetaException 13
2.2.4.5 NoSuchObjectException 13

3 Protocol Details 14
3.1 Common Details 14
3.1.1 Abstract Data Model 14
3.1.2 Timers 14
3.1.3 Initialization 14
3.1.4 Higher-Layer Triggered Events 14
3.1.5 Message Processing Events and Sequencing Rules 14
3.1.5.1 Request Body and Response Body 15
3.1.5.2 Payload 16
3.1.5.3 get_all_databases 16
3.1.5.3.1 Request Body 16
3.1.5.3.2 Response Body 16
3.1.5.3.3 Processing Details 16
3.1.5.4 get_databases 16
3.1.5.4.1 Request Body 17
3.1.5.4.2 Response Body 17
3.1.5.4.3 Processing Details 17
3.1.5.5 get_database 17
3.1.5.5.1 Request Body 17
3.1.5.5.2 Response Body 18
3.1.5.5.3 Processing Details 18
3.1.5.6 get_all_tables 18
3.1.5.6.1 Request Body 18
3.1.5.6.2 Response Body 19
3.1.5.6.3 Processing Details 19
3.1.5.7 get_tables 19

3.1.5.7.1 Request Body 19
3.1.5.7.2 Response Body 20
3.1.5.7.3 Processing Details 20
3.1.5.8 get_table 20
3.1.5.8.1 Request Body 20
3.1.5.8.2 Response Body 21
3.1.5.8.3 Processing Details 21
3.1.5.9 get_tables_by_type 21
3.1.5.9.1 Request Body 21
3.1.5.9.2 Response Body 22
3.1.5.9.3 Processing Details 22
3.1.5.10 get_partition_names 22
3.1.5.10.1 Request Body 22
3.1.5.10.2 Response Body 23
3.1.5.10.3 Processing Details 23
3.1.5.11 get_partitions 23
3.1.5.11.1 Request Body 23
3.1.5.11.2 Response Body 24
3.1.5.11.3 Processing Details 24
3.1.6 Timer Events 24
3.1.7 Other Local Events 24
3.2 Client Details 24

4 Protocol Examples 25
4.1 get_all_databases 25
4.1.1 Request Body 25
4.1.2 Response Body 26
4.2 get_databases 26
4.2.1 Request Body 26
4.2.2 Response Body 26
4.3 get_database 26
4.3.1 Request Body 26
4.3.2 Response Body 27
4.4 get_all_tables 27
4.4.1 Request Body 27
4.4.2 Response Body 28
4.5 get_tables 28
4.5.1 Request Body 28
4.5.2 Response Body 28
4.6 get_table 28
4.6.1 Request Body 28
4.6.2 Response Body 29
4.7 get_tables_by_type 31
4.7.1 Request Body 31
4.7.2 Response Body 32
4.8 get_partition_names 32
4.8.1 Request Body 32
4.8.2 Response Body 32
4.9 get_partitions 32
4.9.1 Request Body 32
4.9.2 Response Body 33

5 Security 38
5.1 Security Considerations for Implementers 38
5.2 Index of Security Parameters 38

6 Appendix A: Thrift Schema 39
6.1 Hive_metastore Thrift Schema 39
6.2 Hive_metastore Thrift APIs 63

7 Appendix B: Product Behavior 64
8 Change Tracking 65
9 Index 67

1 Introduction

The Hive Metastore HTTP protocol specifies a web service API that provides a lightweight interface for clients to read catalog metadata from a **Hive Metastore** database that has been deployed as a data service inside a managed **cluster** environment. This protocol uses an **Apache Thrift** service to provide the specifications to the HTTP method that communicates with the Hive Metastore HTTP service by using the Hive Metastore Thrift API.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

Apache Hadoop: An open-source framework that provides distributed processing of large data sets across clusters of computers that use different programming paradigms and software libraries.

Apache Hive: A data warehousing infrastructure that is based on **Apache Hadoop** and facilitates reading, writing, and managing large datasets that reside in distributed storage and are queried by using SQL syntax.

Apache Knox: A gateway system that provides secure access to data and processing resources in an **Apache Hadoop cluster**.

Apache Thrift: A software framework that enables scalable cross-language development of services that work efficiently between various programming languages.

application: A participant that is responsible for beginning, propagating, and completing an atomic transaction. An application communicates with a transaction manager in order to begin and complete transactions. An application communicates with a transaction manager in order to marshal transactions to and from other applications. An application also communicates in application-specific ways with a resource manager in order to submit requests for work on resources.

Basic: An authentication access type supported by HTTP as defined by [RFC2617].

big data cluster: A grouping of high-value relational data with high-volume big data that provides the computational power of a cluster to increase scalability and performance of **applications**.

cluster: A group of computers that are able to dynamically assign resource tasks among nodes in a group.

Hadoop Distributed File System (HDFS): A core component of **Apache Hadoop**, consisting of a distributed storage and file system that allows files of various formats to be stored across numerous machines or nodes.

Hive Metastore: A database that is external to the **Apache Hive** that stores Hive metadata. The Hive Metastore is responsible for storing table statistics, including table storage location, column names, and table index information.

Interface Definition Language (IDL): The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

JavaScript Object Notation (JSON): A text-based, data interchange format that is used to transmit structured data, typically in Asynchronous JavaScript + XML (AJAX) web applications,

as described in [RFC7159]. The JSON format is based on the structure of ECMAScript (Jscript, JavaScript) objects.

Kerberos: An authentication access type as defined by [RFC1964].

Kubernetes: An open-source container orchestrator that can scale container deployments according to need. Containers are the basic organizational units from which applications on Kubernetes run.

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ApacheHadoop] Apache Software Foundation, "Apache Hadoop", https://hadoop.apache.org/

[ApacheHive] Apache Software Foundation, "Apache Hive", https://hive.apache.org/

[ApacheKnox] Apache Software Foundation, "Apache Knox", https://knox.apache.org/

[ApacheThrift] Apache Software Foundation, "Apache Thrift", https://thrift.apache.org/

[Kubernetes] The Kubernetes Authors, "Kubernetes Documentation", https://kubernetes.io/docs/home/

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.rfc-editor.org/rfc/rfc2818.txt

[RFC7230] Fielding, R., and Reschke, J., Eds., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014, http://www.rfc-editor.org/rfc/rfc7230.txt

[RFC793] Postel, J., Ed., "Transmission Control Protocol: DARPA Internet Program Protocol Specification", RFC 793, September 1981, http://www.rfc-editor.org/rfc/rfc793.txt

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, December 2017, https://www.rfc-editor.org/rfc/rfc8259.txt

1.2.2 Informative References

None.

1.3 Overview

The **Hive Metastore** stores **Apache Hive** [ApacheHive] metadata for databases that are in a **big data cluster**, and the Hive Metastore is exposed out of the big data cluster through the **Apache Knox** [ApacheKnox] access point in the **Apache Hadoop** [ApacheHadoop] **cluster**. The Hive Metastore HTTP protocol specifies APIs that are integrated with HTTP and served through the Knox endpoint.

The Hive Metastore HTTP protocol uses services that are defined and created by using the **Apache Thrift** [ApacheThrift] protocol. For example, the client can be a JavaScript API that uses HTTPS operations as the transport wrapper and Thrift-supported TJSONProtocol, as specified in [ApacheThrift], to encode data by using **JavaScript Object Notation (JSON)**, as specified in [RFC8259], to define and create the Hive Metastore HTTP services.

The Hive Metastore HTTP protocol uses **Basic** or **Kerberos** authentication for all requests.

All requests are initiated by the client, and both the request and response pass through the Knox access point, as illustrated in the following diagram.

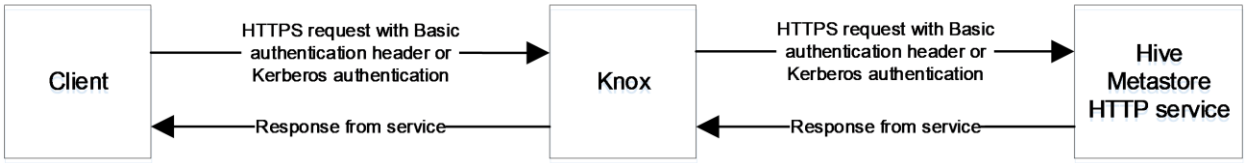

Figure 1: Communication flow

1.4 Relationship to Other Protocols

The Hive Metastore HTTP protocol transmits messages by using HTTPS [RFC7230] [RFC2818] over TCP [RFC793].

The following diagram shows the protocol layering.

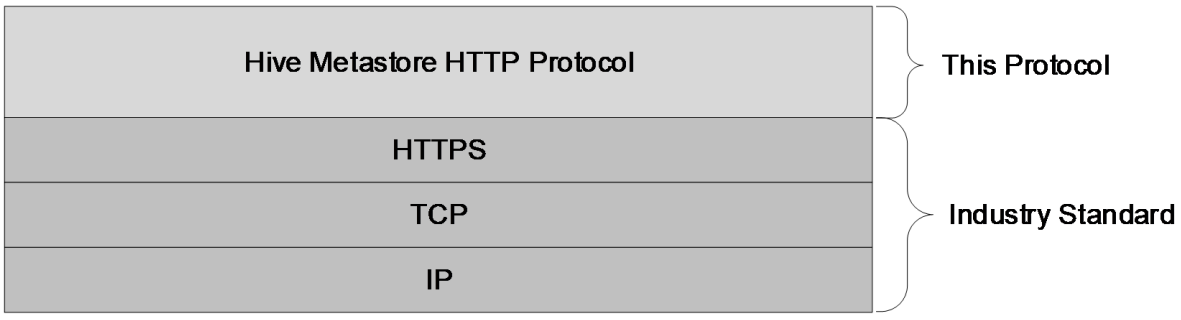

Figure 2: Protocol layering

1.5 Prerequisites/Preconditions

A **Hive Metastore** HTTP service is deployed in a **big data cluster** before the Hive Metastore HTTP protocol can be used.

1.6 Applicability Statement

This protocol supports **Hive Metastore** API calls from a HTTP REST client to the Hive Metastore HTTP service in a **big data cluster**.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The Hive Metastore HTTP Protocol consists of a set of RESTful [REST] web services APIs, and messages MUST use HTTPS over TCP/IP, as specified in [RFC7230], and include Transport Layer Security (TLS) [RFC2818].

2.2 Common Data Types

2.2.1 Namespaces

None.

2.2.2 HTTP Methods

This protocol uses the HTTP operation POST [RFC7230].

2.2.3 URI Parameters

This protocol defines the following common **Uniform Resource Identifier (URI)** parameters.

URI parameter	Description
clusterIp	The IP address of the **cluster**.
knoxPort	The Knox port number of the cluster.
path	The path that is inside the cluster to which Knox redirects the request.

2.2.3.1 clusterIp

The *clusterIp* parameter contains the IP address of the **big data cluster** Knox [ApacheKnox].

2.2.3.2 knoxPort

The *knoxPort* parameter is the endpoint that is exposed by using the **Kubernetes** [Kubernetes] gateway-svc-external service. This port is used to communicate with the **Hive Metastore** server.

2.2.3.3 path

The *path* parameter contains the path that **Apache Knox** [ApacheKnox] needs for mapping to the **Hive Metastore** server in the **big data cluster**.

2.2.4 Data Structures

The following data structures are defined in this protocol.

Data structure	Section	Description
Database	2.2.4.1	Specifies information about the database.

Data structure	Section	Description
Table	2.2.4.2	Specifies information about the table.
Partition	2.2.4.3	Specifies information about the partition.

In addition, this section defines the following exceptions.

Exception	Section	Description
MetaException	2.2.4.4	Defines the exception.
NoSuchObjectException	2.2.4.5	Contains a message that specifies the exception.

The data structures flow through this protocol by using structures that are defined in **Apache Thrift** [ApacheThrift]. This protocol uses the Hive_metastore Thrift schema definition as described in section 6.1 to generate all client communications.

2.2.4.1 Database

The **Database** structure specifies information about the database.

```
struct Database {
  1: string name,
  2: string description,
  3: string locationUri,
  4: map<string, string> parameters, // properties associated with the database
  5: optional PrincipalPrivilegeSet privileges,
  6: optional string ownerName,
  7: optional PrincipalType ownerType
}
```

name: The name of the database.

description: A description of the database.

locationUri: The **URI** where the database is located.

parameters: A field to store a list of name and value pairs associated with the database, such as a comment or other user-level parameters.

privileges: A list of user, group, and role permissions for the database.

ownerName: The name for the database owner.

ownerType: The type of ownerName. Valid values are user, group, or role.

2.2.4.2 Table

The **Table** structure specifies information about the table.

```
struct Table {
  1: string tableName,                // name of the table
  2: string dbName,                   // database name ('default')
  3: string owner,                    // owner of this table
  4: i32    createTime,               // creation time of the table
  5: i32    lastAccessTime,           // last access time (usually this will be filled from
HDFS and shouldn't be relied on)
  6: i32    retention,                // retention time
```

```
  7: StorageDescriptor sd,            // storage descriptor of the table
  8: list<FieldSchema> partitionKeys, // partition keys of the table. only primitive types
are supported
  9: map<string, string> parameters,   // to store comments or any other user level
parameters
  10: string viewOriginalText,         // original view text, null for non-view
  11: string viewExpandedText,         // expanded view text, null for non-view
  12: string tableType,                // table type enum, e.g. EXTERNAL_TABLE
  13: optional PrincipalPrivilegeSet privileges,
  14: optional bool temporary=false,
  15: optional bool rewriteEnabled     // rewrite enabled or not
}
```

tableName: The name of the table.

dbName: The name of the database.

owner: The owner of the table.

createTime: The time that the table was created.

lastAccessTime: The last time that the table was accessed. This field can be populated from **Hadoop Distributed File System (HDFS)** [ApacheHadoop] and can be inaccurate.

retention: The amount of time that the table is retained.

sd: The storage descriptor of the table.

partitionKeys: The partition keys of the table. Only primitive types are supported.

parameters: A field to store comments or any other user-level parameters.

viewOriginalText: View the original text of the table. The value is null if no text is to be displayed.

viewExpandedText: View the full text of the table. The value is null if no text is to be displayed.

tableType: The type of the table, such as EXTERNAL_TABLE.

privileges: A list of user, group, and role permissions for the table.

temporary: A Boolean that is set to true if the table is temporary and false if the table is permanent.

rewriteEnabled: A Boolean that specifies whether the table can be rewritten.

2.2.4.3 Partition

The **Partition** structure specifies information about the partition.

```
struct Partition {
  1: list<string> values, // string value is converted to appropriate partition key type
  2: string       dbName,
  3: string       tableName,
  4: i32          createTime,
  5: i32          lastAccessTime,
  6: StorageDescriptor   sd,
  7: map<string, string> parameters,
  8: optional PrincipalPrivilegeSet privileges
}
```

values: The string values that are converted to the appropriate partition key types.

dbName: The name of the database.

tableName: The name of the table.

createTime: The time that the table was created.

lastAccessTime: The last time that the table was accessed. This field can be populated from **HDFS** [ApacheHadoop] and can be inaccurate.

sd: The storage descriptor of the table.

parameters: A field in which to store comments and any other user-level parameters.

privileges: A list of user, group, and role permissions for the partition.

2.2.4.4 MetaException

The **MetaException** exception defines the exception.

```
exception MetaException {
  1: string message
}
```

message: The message that specifies the exception.

2.2.4.5 NoSuchObjectException

The **NoSuchObjectException** exception contains a message that specifies the exception.

```
exception NoSuchObjectException {
  1: string message
}
```

message: The message that specifies the exception.

3 Protocol Details

3.1 Common Details

This section specifies protocol details that are common to multiple roles.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation can maintain to participate in this protocol. The organization is provided to help explain how the protocol works. This document does not require that implementations adhere to this model, provided that the external behavior of the implementation is consistent with that specified in this document.

The following elements can be queried by this protocol:

- Database (sections 3.1.5.3 through 3.1.5.5)
- Table (sections 3.1.5.6 through 3.1.5.9)
- Partition (sections 3.1.5.10 through 3.1.5.11)

3.1.2 Timers

None.

3.1.3 Initialization

For a client to use the protocol, the client MUST have a **big data cluster** that is running in the **Apache Hadoop** [ApacheHadoop] **cluster**, and a **Hive Metastore** HTTP service MUST be running on the big data cluster.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

The **Hive Metastore** database is queried by using this protocol.

The HTTP POST operation sends a query by using the Hive_metastore **Thrift** schema, as specified in the request and response sections for each method. These methods are the same methods that appear in the full Hive_metastore Thrift schema in section 6.1.

The following methods can be performed during the HTTP POST operation on the Hive Metastore database.

Method	Section	Description
get_all_databases	3.1.5.3	Retrieves a list of the names of all the databases.
get_databases	3.1.5.4	Retrieves a list of database names that match a specified pattern.
get_database	3.1.5.5	Retrieves information about a specified database.
get_all_tables	3.1.5.6	Retrieves a list of the names of all the tables in a specified database.

Method	Section	Description
get_tables	3.1.5.7	Retrieves a list of the table names that match a specified pattern in a specified database.
get_table	3.1.5.8	Retrieves information about a specified table.
get_tables_by_type	3.1.5.9	Retrieves a list of the names of all the tables with names that match a specified pattern and with a table type that matches a specified type in a specified database.
get_partition_names	3.1.5.10	Retrieves a list of partition names.
get_partitions	3.1.5.11	Retrieves a list of partitions.

A Hive Metastore Thrift HTTP client is created by using the Apache Thrift [ApacheThrift] toolchain to query the Hive Metastore Thrift API. With this API, the payload and request are automatically generated by Thrift.

The method is invoked by sending a POST operation to the following **URI**.

```
https://<clusterIp>:<knoxPort>/<path>
```

See section 2.2.3 for a description of the *clusterIp*, *knoxPort*, and *path* parameters.

The client that invokes the method MUST have a **big data cluster** hostname and a big data cluster Knox port number and service path and MUST use **Basic** or **Kerberos** authentication.

3.1.5.1 Request Body and Response Body

The request body and the response body are in TJSONProtocol format [ApacheThrift].

The following are the positions and their descriptions of the parameters that correspond to fields in a request body or response body array.

Position	Parameter	Description
1	version	An integer that provides the version of the schema to be used. The value of the parameter MUST be 1.
2	message name	A string that provides the name of the message.
3	message type	An integer that provides the type of the message. The codes and their corresponding message types are as follows: ▪ 1 = Call: A request sent by a client to the server. ▪ 2 = Reply: A response sent by the server to the client. The response can be an exception that is defined in the Thrift **Interface Definition Language (IDL)** file. ▪ 3 = Exception: A response sent by the server to the client if the server encounters an error that is not defined in the Thrift IDL file. ▪ 4 = Oneway: A request sent by a client to the server for which no response is sent or expected.
4	seqid	An integer that provides the sequence ID of the message.
5	payload	A **JSON** object that provides the request or response payload.

3.1.5.2 Payload

The request body and the response body payload values are in TJSONProtocol format [ApacheThrift].

Each entry is encoded with position, parameter, and value.

3.1.5.3 get_all_databases

The get_all_databases method retrieves a list of the names of all the databases in the **Hive Metastore**.

3.1.5.3.1 Request Body

The get_all_databases request body is a **JSON** object as specified in section 2.2.4.1. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_all_databases"
3	message type	1
4	seqid	1
5	payload	{}

In the get_all_databases request body, the payload is an empty value.

For more information, see the example of the get_all_databases request body in section 4.1.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.3.2 Response Body

The get_all_databases response body contains all the database names as a list of strings.

The response message for the get_all_databases method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and all the databases were returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_all_databases response body in section 4.1.2.

3.1.5.3.3 Processing Details

The get_all_databases method queries the **Hive Metastore** to return a list of all the database names.

3.1.5.4 get_databases

The get_databases method retrieves a list of the database names in the **Hive Metastore** that match a specified pattern.

3.1.5.4.1 Request Body

The get_databases request body is a **JSON** object as specified in section 2.2.4.1. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_databases"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_databases method.

Position	Parameter	Description
1	pattern	Specifies the pattern of the database name for which to search.

For more information, see the example of the get_databases request body in section 4.2.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.4.2 Response Body

The get_databases response body is a list of strings of the database names that match a specified pattern.

The response message for the get_databases method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the information was returned for the databases that match the specified pattern.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_databases response body in section 4.2.2.

3.1.5.4.3 Processing Details

The get_databases method queries the **Hive Metastore** to return a list of database names that match a specified pattern.

3.1.5.5 get_database

The get_database method queries the databases in the **Hive Metastore** by database name.

3.1.5.5.1 Request Body

The get_database request body is a **JSON** object as specified in section 2.2.4.1. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_database"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_database method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search

For more information, see the example of the get_database request body in section 4.3.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.5.2 Response Body

The get_database response body is a **Database** data structure that is defined in section 2.2.4.1.

The response message for the get_database method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the information was returned about the databases by database name.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_database response body in section 4.3.2.

3.1.5.5.3 Processing Details

The get_database method queries the databases in the **Hive Metastore** by database name.

3.1.5.6 get_all_tables

The get_all_tables method retrieves a list of the names of all the tables in the database in the **Hive Metastore** that is specified by a database name.

3.1.5.6.1 Request Body

The get_all_tables request body is a **JSON** object as specified in section 2.2.4.2. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_all_tables"
3	message type	1

Position	Parameter	Value
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_all_tables method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search

For more information, see the example of the get_all_tables request body in section 4.4.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.6.2 Response Body

The get_all_tables response body is a list of the names of all the tables in a specified database.

The response message for the get_all_tables method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and all the tables in the databases specified by name were returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_all_tables response body in section 4.4.2.

3.1.5.6.3 Processing Details

The get_all_tables method queries the **Hive Metastore** to return a list of the names of all the tables in the database that is specified by a database name.

3.1.5.7 get_tables

The get_tables method retrieves a list of the table names that match a specified pattern in a specified database in the **Hive Metastore**.

3.1.5.7.1 Request Body

The get_tables request body is a **JSON** object as specified in section 2.2.4.2. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_tables"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_tables method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search for table names.
2	pattern	Specifies a variable value that consists of a string with wildcards indicated by asterisks '*'.

For more information, see the example of the get_tables request body in section 4.5.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.7.2 Response Body

The get_tables response body is a list of the table names that match a specified pattern in a specified database.

The response message for the get_tables method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the tables whose names match the pattern in the specified database were returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_tables response body in section 4.5.2.

3.1.5.7.3 Processing Details

The get_tables method queries the **Hive Metastore** to return a list of the table names that match a specified pattern in a specified database.

3.1.5.8 get_table

The get_table method retrieves information about a specified table in a specified database in the **Hive Metastore**.

3.1.5.8.1 Request Body

The get_table request body is a **JSON** object as specified in section 2.2.4.2. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_table"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_tables method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search for the specified table.
2	tbl_name	Specifies the name of the table to search.

For more information, see the example of the get_table request body in section 4.6.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.8.2 Response Body

The get_table response body is a **Table** data structure that is defined in section 2.2.4.2.

The response message for the get_table method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the information about the specified table in the specified database was returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_table response body in section 4.6.2.

3.1.5.8.3 Processing Details

The get_table method queries the **Hive Metastore** to return information about a specified table in a specified database.

3.1.5.9 get_tables_by_type

The get_tables_by_type method retrieves a list of the names of all the tables with names that match a specified pattern and with a table type that matches a specified type in a specified database in the **Hive Metastore**.

3.1.5.9.1 Request Body

The get_tables_by_type request body is a **JSON** object as specified in section 2.2.4.2. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_tables_by_type"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_tables_by_type method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search for tables.

Position	Parameter	Description
2	tbl_name	Specifies the pattern of the table names to search.
3	tbl_type	Specifies the type of table for which to search.

For more information, see the example of the get_tables_by_type request body in section 4.7.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.9.2 Response Body

The get_tables_by_type response body is a list of table names.

The response message for the get_tables_by_type method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the list of tables whose names match the specified pattern and whose table types match the tableType in the specified database was returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_tables_by_type response body in section 4.7.2.

3.1.5.9.3 Processing Details

The get_tables_by_type method queries the **Hive Metastore** to return a list of the names of all the tables with names that match a specified pattern and with a table type that matches a specified type in a specified database.

3.1.5.10 get_partition_names

The get_partition_names method retrieves a list of partition names of a specified table in a specified database in the **Hive Metastore**.

3.1.5.10.1 Request Body

The get_partition_names request body is a **JSON** object as specified in section 2.2.4.3. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_partition_names"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_partition_names method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search for partition names.
2	tbl_name	Specifies the name of the table to search for partition names.
3	max_parts	Specifies the maximum number of partition names to return.

For more information, see the example of the get_partition_names request body in section 4.8.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.10.2 Response Body

The get_partition_names response body is a list of names of partitions of a specified table in a specified database.

The response message for the get_partition_names method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the list of partition names of the specified table in the specified database was returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_partition_names response body in section 4.8.2.

3.1.5.10.3 Processing Details

The get_partition_names method queries the **Hive Metastore** to return a list of partition names of a specified table in a specified database.

3.1.5.11 get_partitions

The get_partitions method retrieves a list of partitions of a specified table in a specified database in the **Hive Metastore**.

3.1.5.11.1 Request Body

The get_partitions request body is a **JSON** object as specified in section 2.2.4.3. It is encoded with the values that are defined in the following table.

Position	Parameter	Value
1	version	1
2	message name	"get_partitions"
3	message type	1
4	seqid	1
5	payload	Variable value defined below.

The following are the values of the request payload for the get_partitions method.

Position	Parameter	Description
1	db_name	Specifies the name of the database to search for partitions.
2	tbl_name	Specifies the name of the table to search for partitions.
3	max_parts	Specifies the maximum number of partitions to return.

For more information, see the example of the get_partitions request body in section 4.9.1 and the **Hive Metastore Thrift** API signature [ApacheThrift] in section 6.2.

3.1.5.11.2 Response Body

The get_partitions response body is a list of **Partition** data structures that are described in section 2.2.4.3.

The response message for the get_partitions method can result in the following status codes.

HTTP status code	Description
200	The query was accepted, and the list of partitions of the specified table in the specified database was returned.
401	Unauthorized. The **Basic** or **Kerberos** authentication failed, and the client does not have the correct certificate.

For more information, see the example of the get_partitions response body in section 4.9.2.

3.1.5.11.3 Processing Details

The get_partitions method queries the **Hive Metastore** to return a list of partitions of a specified table in a specified database.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Client Details

The client role of this protocol is simply a pass-through and requires no additional timers or other state. Calls made by the higher-layer protocol or **application** are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

4 Protocol Examples

This section presents examples of the parts of a request, such as **URI**, request, and request header. Where necessary, the example request and response bodies throughout this section use "hmshttptestdatabase" as the name of the database.

The following is an example of a URI.

```
'https://ClusterIP:30443/gateway/default/hmshttpthrift/api/hms';
```

The following is an example of a request that can be sent to the included URI.

```
curl -k POST -u admin:<adminPassword> --data '[request_body_data]' -H "Content-Type:
application/vnd.apache.thrift.json"
https://<clusterIp>:<knoxPort>/gateway/default/hmshttpthrift/api/hms
```

The following is an example of a request header. There are no common HTTP headers defined by this protocol.

```
POST https://clusterIP:30443/gateway/default/hmshttpthrift/api/hms
Connection: close
Authorization: Basic cm9vdDpZdWtvbjkwMA==
Content-length: 72
Content-Type: application/x-thrift
Host: 10.91.88.177:30443
```

4.1 get_all_databases

This example illustrates how the client might obtain a list of the names of all the databases in the **Hive Metastore** by using the get_all_databases method (see section 3.1.5.3).

4.1.1 Request Body

The following is an example of a get_all_databases request body that is defined in section 3.1.5.3.1.

```
[1,"get_all_databases",1,1,{}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_all_databases"
3	message type	1
4	seqid	1
5	payload	{}

4.1.2 Response Body

The following is an example of a get_all_databases HTTP response body that is defined in section 3.1.5.3.2.

```
[1,"get_all_databases",2,1,{"0":{"lst":["str",2,"default","hmshttptestdatabase"]}}]
```

4.2 get_databases

This example illustrates how the client might obtain a list of database names in the **Hive Metastore** that match a specified pattern by using the get_databases method (see section 3.1.5.4).

4.2.1 Request Body

The following is an example of a get_databases request body that is defined in section 3.1.5.4.1.

```
[1,"get_databases",1,1,{"1":{"str":"default*"}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_databases"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":"default*"}}

4.2.2 Response Body

The following is an example of a get_databases HTTP response body that is defined in section 3.1.5.4.2.

```
[1,"get_databases",2,1,{"0":{"lst":["str",1,"default"]}}]
```

4.3 get_database

This example illustrates how the client might query the databases in the **Hive Metastore** by database name by using the get_database method (see section 3.1.5.5).

4.3.1 Request Body

The following is an example of a get_database request body that is defined in section 3.1.5.5.1.

```
[1,"get_database",1,1,{"1":{"str":"default"}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_database"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":"default"}}

4.3.2 Response Body

The following is an example of a get_database HTTP response body that is defined in section 3.1.5.5.2.

```
[1,"get_database",2,1,{"0":{"rec":{"1":{"str":"default"},"2":{"str":"Default Hive
database"},"3":{"str":"hdfs://nmnode-0-0.nmnode-0-
svc:9000/user/hive/warehouse"},"4":{"map":["str","str",0,{}]},"6":{"str":"public"},"7":{"
i32":2}}}}]
```

4.4 get_all_tables

This example illustrates how the client might obtain a list of the names of all the tables in a database in the **Hive Metastore** that is specified by database name by using the get_all_tables method (see section 3.1.5.6).

4.4.1 Request Body

The following is an example of a get_all_tables request body that is defined in section 3.1.5.6.1.

```
[1,"get_all_tables",1,1,{"1":{"str":"default"}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_all_tables"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":"default"}}

4.4.2 Response Body

The following is an example of a get_all_tables HTTP response body that is defined in section 3.1.5.6.2.

```
[1,"get_all_tables",2,1,{"0":{"lst":["str",0]}}]
```

4.5 get_tables

This example illustrates how the client might obtain a list of the table names that match a specified pattern in a specified database in the **Hive Metastore** by using the get_tables method (see section 3.1.5.7).

4.5.1 Request Body

The following is an example of a get_tables request body that is defined in section 3.1.5.5.2.

```
[1,"get_tables",1,1,{"1":{"str":"default"},"2":{"str":"*"}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_tables"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":"default"},"2":{"str":"*"}}

4.5.2 Response Body

The following is an example of a get_tables HTTP response body that is defined in section 3.1.5.7.2.

```
[1,"get_tables",2,1,{"0":{"lst":["str",0]}}]
```

4.6 get_table

This example illustrates how the client might obtain information about a specified table in a specified database in the **Hive Metastore** by using the get_table method (see section 3.1.5.8).

4.6.1 Request Body

The following is an example of a get_table request body that is defined in section 3.1.5.8.1.

```
[1,"get_table",1,1,{"1":{"str":"hmshttptestdatabase"},"2":{"str":"test_table"}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_table"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":"hmshttptestdatabase"},"2":{"str":"test_table"}}

4.6.2 Response Body

The following is an example of a get_table HTTP response body that is defined in section 3.1.5.8.2.

```
[1, "get table", 2, 1, {
        "0": {
            "rec": {
                "1": {
                    "str": "test table"
                },
                "2": {
                    "str": "hmshttptestdatabase"
                },
                "3": {
                    "str": "root"
                },
                "4": {
                    "i32": 1566250831
                },
                "5": {
                    "i32": 0
                },
                "6": {
                    "i32": 0
                },
                "7": {
                    "rec": {
                        "1": {
                            "lst": ["rec", 2, {
                                    "1": {
                                        "str": "name"
                                    },
                                    "2": {
                                        "str": "string"
                                    }
                                }, {
                                    "1": {
                                        "str": "age"
                                    },
                                    "2": {
                                        "str": "int"
                                    }
                                }
                            ]
                        },
                        "2": {
                            "str": "hdfs://nmnode-0-0.nmnode-0-
svc:9000/hmshttptest/warehouse/hmshttptestdatabase/test_table"
                        },
                        "3": {
```

```
                            "str": "org.apache.hadoop.mapred.SequenceFileInputFormat"
                        },
                        "4": {
                            "str":
"org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat"
                        },
                        "5": {
                            "tf": 0
                        },
                        "6": {
                            "i32": -1
                        },
                        "7": {
                            "rec": {
                                "2": {
                                    "str":
"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe"
                                },
                                "3": {
                                    "map": ["str", "str", 1, {
                                            "serialization.format": "1"
                                        }
                                    ]
                                }
                            }
                        },
                        "8": {
                            "lst": ["str", 0]
                        },
                        "9": {
                            "lst": ["rec", 0]
                        },
                        "10": {
                            "map": ["str", "str", 0, {}
                            ]
                        },
                        "11": {
                            "rec": {
                                "1": {
                                    "lst": ["str", 0]
                                },
                                "2": {
                                    "lst": ["lst", 0]
                                },
                                "3": {
                                    "map": ["lst", "str", 0, {}
                                    ]
                                }
                            }
                        },
                        "12": {
                            "tf": 0
                        }
                    }
                },
                "8": {
                    "lst": ["rec", 1, {
                            "1": {
                                "str": "hair color"
                            },
                            "2": {
                                "str": "string"
                            }
                        }
                    ]
                },
                "9": {
                    "map": ["str", "str", 11, {
                            "owner": "testuser",
```

```
                            "spark.sql.sources.schema.numPartCols": "1",
                            "spark.sql.sources.schema.part.0":
"{\"type\":\"struct\",\"fields\":[{\"name\":\"name\",\"type\":\"string\",\"nullable\":true,\"
metadata\":{}},{\"name\":\"age\",\"type\":\"integer\",\"nullable\":true,\"metadata\":{}},{\"n
ame\":\"hair_color\",\"type\":\"string\",\"nullable\":true,\"metadata\":{}}]}",
                            "spark.sql.sources.schema.partCol.0": "hair color",
                            "transient_lastDdlTime": "1566250843",
                            "spark.sql.statistics.numRows": "2",
                            "comment": "Table Comment",
                            "spark.sql.sources.schema.numParts": "1",
                            "spark.sql.create.version": "2.4.64778",
                            "spark.sql.statistics.totalSize": "216",
                            "status": "staging"
                        }
                    ]
                },
                "12": {
                    "str": "MANAGED_TABLE"
                },
                "15": {
                    "tf": 0
                }
            }
        }
    }
]
```


4.7 get_tables_by_type

This example illustrates how the client might obtain a list of the names of all the tables with names that match a specified pattern and with a table type that matches a specified type in a specified database in the **Hive Metastore** by using the get_tables_by_type method (see section 3.1.5.9).

4.7.1 Request Body

The following is an example of a get_tables_by_type request body that is defined in section 3.1.5.9.1.


```
[1,"get_tables_by_type",1,1,{"1":{"str":"hmshttptestdatabase"},"2":{"str":".*"},"3":{"str":"M
ANAGED_TABLE"}}]
```


The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_tables_by_type"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":"hmshttptestdatabase"},"2":{"str":".*"},"3":{"str":"MANAGED_TABLE"}}

4.7.2 Response Body

The following is an example of a get_tables_by_type HTTP response body that is defined in section 3.1.5.9.2.

```
[1,"get_tables_by_type",2,1,{"0":{"lst":["str",1,"test_table"]}}]
```

4.8 get_partition_names

This example illustrates how the client might obtain a list of partition names of a specified table in a specified database in the **Hive Metastore** by using the get_partition_names method (see section 3.1.5.10).

4.8.1 Request Body

The following is an example of a get_partition_names request body that is defined in section 3.1.5.10.1.

```
[1,"get_partition_names",1,1,{"1":{"str":"hmshttptestdatabase"},"2":{"str":"test_table"},"3":
{"i16":10}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_partition_names"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":" hmshttptestdatabase"},"2":{"str":"test_table"},"3":{"i16":10}}

4.8.2 Response Body

The following is an example of a get_partition_names HTTP response body that is defined in section 3.1.5.10.2.

```
[1,"get_partition_names",2,1,{"0":{"lst":["str",2,"hair_color=black","hair_color=brown"]}}]
```

4.9 get_partitions

This example illustrates how the client might obtain a list of partitions of a specified table in a specified database in the **Hive Metastore** by using the get_partitions method (see section 3.1.5.11).

4.9.1 Request Body

The following is an example of a get_partitions request body that is defined in section 3.1.5.11.1.

```
[1,"get_partitions",1,1,{"1":{"str":"hmshttptestdatabase"},"2":{"str":"test_table"},"3":{"i16
":10}}]
```

The parameters of the request body are interpreted as follows.

Position	Parameter	Value
1	version	1
2	message name	"get_partitions"
3	message type	1
4	seqid	1
5	payload	{"1":{"str":" hmshttptestdatabase"},"2":{"str":"test_table"},"3":{"i16":10}}

4.9.2 Response Body

The following is an example of a get_partitions HTTP response body that is defined in section 3.1.5.11.2.

```
[
 1,
 "get partitions",
 2,
 1,
 {
   "0": {
     "lst": [
       "rec",
       2,
       {
         "1": {
           "lst": [
             "str",
             1,
             "black"
           ]
         },
         "2": {
           "str": "hmshttptestdatabase"
         },
         "3": {
           "str": "test_table"
         },
         "4": {
           "i32": 1566250836
         },
         "5": {
           "i32": 0
         },
         "6": {
           "rec": {
             "1": {
               "lst": [
                 "rec",
                 2,
                 {
                   "1": {
                     "str": "name"
```

```
                    },
                    "2": {
                      "str": "string"
                    }
                  },
                  {
                    "1": {
                      "str": "age"
                    },
                    "2": {
                      "str": "int"
                    }
                  }
                ]
              },
              "2": {
                "str": "hdfs://nmnode-0-0.nmnode-0-
svc:9000/hmshttptest/warehouse/hmshttptestdatabase/test_table/hair_color=black"
              },
              "3": {
                "str": "org.apache.hadoop.mapred.SequenceFileInputFormat"
              },
              "4": {
                "str": "org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat"
              },
              "5": {
                "tf": 0
              },
              "6": {
                "i32": -1
              },
              "7": {
                "rec": {
                  "2": {
                    "str": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe"
                  },
                  "3": {
                    "map": [
                      "str",
                      "str",
                      1,
                      {
                        "serialization.format": "1"
                      }
                    ]
                  }
                }
              },
              "8": {
                "lst": [
                  "str",
                  0
                ]
              },
              "9": {
                "lst": [
                  "rec",
                  0
                ]
              },
              "10": {
                "map": [
                  "str",
                  "str",
                  0,
                  {}
                ]
              },
              "11": {
```

```
          "rec": {
            "1": {
              "lst": [
                "str",
                0
              ]
            },
            "2": {
              "lst": [
                "lst",
                0
              ]
            },
            "3": {
              "map": [
                "lst",
                "str",
                0,
                {}
              ]
            }
          }
        },
        "12": {
          "tf": 0
        }
      }
    },
    "7": {
      "map": [
        "str",
        "str",
        6,
        {
          "transient_lastDdlTime": "1566250836",
          "totalSize": "108",
          "numRows": "0",
          "rawDataSize": "0",
          "COLUMN_STATS_ACCURATE": "{\"BASIC_STATS\":\"true\"}",
          "numFiles": "1"
        }
      ]
    }
  },
  {
    "1": {
      "lst": [
        "str",
        1,
        "brown"
      ]
    },
    "2": {
      "str": "hmshttptestdatabase"
    },
    "3": {
      "str": "test table"
    },
    "4": {
      "i32": 1566250838
    },
    "5": {
      "i32": 0
    },
    "6": {
      "rec": {
        "1": {
          "lst": [
            "rec",
```

```
                    2,
                    {
                      "1": {
                        "str": "name"
                      },
                      "2": {
                        "str": "string"
                      }
                    },
                    {
                      "1": {
                        "str": "age"
                      },
                      "2": {
                        "str": "int"
                      }
                    }
                  ]
                },
                "2": {
                  "str": "hdfs://nmnode-0-0.nmnode-0-
svc:9000/hmshttptest/warehouse/hmshttptestdatabase/test_table/hair_color=brown"
                },
                "3": {
                  "str": "org.apache.hadoop.mapred.SequenceFileInputFormat"
                },
                "4": {
                  "str": "org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat"
                },
                "5": {
                  "tf": 0
                },
                "6": {
                  "i32": -1
                },
                "7": {
                  "rec": {
                    "2": {
                      "str": "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe"
                    },
                    "3": {
                      "map": [
                        "str",
                        "str",
                        1,
                        {
                          "serialization.format": "1"
                        }
                      ]
                    }
                  }
                },
                "8": {
                  "lst": [
                    "str",
                    0
                  ]
                },
                "9": {
                  "lst": [
                    "rec",
                    0
                  ]
                },
                "10": {
                  "map": [
                    "str",
                    "str",
                    0,
```

```
                {}
              ]
            },
            "11": {
              "rec": {
                "1": {
                  "lst": [
                    "str",
                    0
                  ]
                },
                "2": {
                  "lst": [
                    "lst",
                    0
                  ]
                },
                "3": {
                  "map": [
                    "lst",
                    "str",
                    0,
                    {}
                  ]
                }
              }
            },
            "12": {
              "tf": 0
            }
          }
        },
        "7": {
          "map": [
            "str",
            "str",
            6,
            {
              "transient_lastDdlTime": "1566250838",
              "totalSize": "108",
              "numRows": "0",
              "rawDataSize": "0",
              "COLUMN_STATS_ACCURATE": "{\"BASIC_STATS\":\"true\"}",
              "numFiles": "1"
            }
          ]
        }
      }
    ]
  }
]
```

5 Security

5.1 Security Considerations for Implementers

Basic authentication and **Kerberos** authentication are the only supported authentication methods.

5.2 Index of Security Parameters

None.

6 Appendix A: Thrift Schema

6.1 Hive_metastore Thrift Schema

This section presents the **Apache Thrift** file that is used to generate and test a JavaScript API.

The Open source Reference Thrift file is found at:
https://raw.githubusercontent.com/apache/hive/rel/release-2.3.0/metastore/if/hive_metastore.thrift

The following code shows the contents of the file.

```
#!/usr/local/bin/thrift -java

/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#
# Thrift Service that the MetaStore is built on
#

include "share/fb303/if/fb303.thrift"

namespace java org.apache.hadoop.hive.metastore.api
namespace php metastore
namespace cpp Apache.Hadoop.Hive

const string DDL_TIME = "transient_lastDdlTime"

struct Version {
  1: string version,
  2: string comments
}

struct FieldSchema {
  1: string name, // name of the field
  2: string type, // type of the field. primitive types defined above, specify
list<TYPE_NAME>, map<TYPE_NAME, TYPE_NAME> for lists & maps
  3: string comment
}

struct SQLPrimaryKey {
  1: string table db,    // table schema
  2: string table_name,  // table name
  3: string column_name, // column name
  4: i32 key_seq,        // sequence number within primary key
  5: string pk name,     // primary key name
  6: bool enable_cstr,   // Enable/Disable
  7: bool validate cstr,  // Validate/No validate
  8: bool rely_cstr      // Rely/No Rely
}
```

```
struct SQLForeignKey {
  1: string pktable_db,    // primary key table schema
  2: string pktable_name,  // primary key table name
  3: string pkcolumn_name, // primary key column name
  4: string fktable_db,    // foreign key table schema
  5: string fktable name,  // foreign key table name
  6: string fkcolumn_name, // foreign key column name
  7: i32 key_seq,          // sequence within foreign key
  8: i32 update_rule,      // what happens to foreign key when parent key is updated
  9: i32 delete_rule,      // what happens to foreign key when parent key is deleted
  10: string fk name,      // foreign key name
  11: string pk_name,      // primary key name
  12: bool enable_cstr,    // Enable/Disable
  13: bool validate_cstr,  // Validate/No validate
  14: bool rely_cstr       // Rely/No Rely
}

struct Type {
  1: string          name,             // one of the types in PrimitiveTypes or
CollectionTypes or User defined types
  2: optional string type1,            // object type if the name is 'list' (LIST_TYPE), key
type if the name is 'map' (MAP_TYPE)
  3: optional string type2,            // val type if the name is 'map' (MAP TYPE)
  4: optional list<FieldSchema> fields // if the name is one of the user defined types
}

enum HiveObjectType {
  GLOBAL = 1,
  DATABASE = 2,
  TABLE = 3,
  PARTITION = 4,
  COLUMN = 5,
}

enum PrincipalType {
  USER = 1,
  ROLE = 2,
  GROUP = 3,
}

const string HIVE_FILTER_FIELD_OWNER = "hive_filter_field_owner__"
const string HIVE FILTER FIELD PARAMS = "hive filter field params  "
const string HIVE_FILTER_FIELD_LAST_ACCESS = "hive_filter_field_last_access__"

enum PartitionEventType {
  LOAD DONE = 1,
}

// Enums for transaction and lock management
enum TxnState {
    COMMITTED = 1,
    ABORTED = 2,
    OPEN = 3,
}

enum LockLevel {
    DB = 1,
    TABLE = 2,
    PARTITION = 3,
}

enum LockState {
    ACQUIRED = 1,       // requester has the lock
    WAITING = 2,        // requester is waiting for the lock and should call checklock at a
later point to see if the lock has been obtained.
    ABORT = 3,          // the lock has been aborted, most likely due to timeout
    NOT_ACQUIRED = 4,   // returned only with lockNoWait, indicates the lock was not
available and was not acquired
}
```

```
enum LockType {
    SHARED_READ = 1,
    SHARED_WRITE = 2,
    EXCLUSIVE = 3,
}

enum CompactionType {
    MINOR = 1,
    MAJOR = 2,
}

enum GrantRevokeType {
    GRANT = 1,
    REVOKE = 2,
}

enum DataOperationType {
    SELECT = 1,
    INSERT = 2
    UPDATE = 3,
    DELETE = 4,
    UNSET = 5,//this is the default to distinguish from NULL from old clients
    NO_TXN = 6,//drop table, insert overwrite, etc - something non-transactional
}

// Types of events the client can request that the metastore fire.  For now just support DML
operations, as the metastore knows
// about DDL operations and there's no reason for the client to request such an event.
enum EventRequestType {
    INSERT = 1,
    UPDATE = 2,
    DELETE = 3,
}

struct HiveObjectRef{
  1: HiveObjectType objectType,
  2: string dbName,
  3: string objectName,
  4: list<string> partValues,
  5: string columnName,
}

struct PrivilegeGrantInfo {
  1: string privilege,
  2: i32 createTime,
  3: string grantor,
  4: PrincipalType grantorType,
  5: bool grantOption,
}

struct HiveObjectPrivilege {
  1: HiveObjectRef  hiveObject,
  2: string principalName,
  3: PrincipalType principalType,
  4: PrivilegeGrantInfo grantInfo,
}

struct PrivilegeBag {
  1: list<HiveObjectPrivilege> privileges,
}

struct PrincipalPrivilegeSet {
  1: map<string, list<PrivilegeGrantInfo>> userPrivileges, // user name -> privilege grant
info
  2: map<string, list<PrivilegeGrantInfo>> groupPrivileges, // group name -> privilege grant
info
  3: map<string, list<PrivilegeGrantInfo>> rolePrivileges, //role name -> privilege grant
info
```

```
}

struct GrantRevokePrivilegeRequest {
  1: GrantRevokeType requestType;
  2: PrivilegeBag privileges;
  3: optional bool revokeGrantOption;  // Only for revoke request
}

struct GrantRevokePrivilegeResponse {
  1: optional bool success;
}

struct Role {
  1: string roleName,
  2: i32 createTime,
  3: string ownerName,
}

// Representation of a grant for a principal to a role
struct RolePrincipalGrant {
  1: string roleName,
  2: string principalName,
  3: PrincipalType principalType,
  4: bool grantOption,
  5: i32 grantTime,
  6: string grantorName,
  7: PrincipalType grantorPrincipalType
}

struct GetRoleGrantsForPrincipalRequest {
  1: required string principal name,
  2: required PrincipalType principal_type
}

struct GetRoleGrantsForPrincipalResponse {
  1: required list<RolePrincipalGrant> principalGrants;
}

struct GetPrincipalsInRoleRequest {
  1: required string roleName;
}

struct GetPrincipalsInRoleResponse {
  1: required list<RolePrincipalGrant> principalGrants;
}

struct GrantRevokeRoleRequest {
  1: GrantRevokeType requestType;
  2: string roleName;
  3: string principalName;
  4: PrincipalType principalType;
  5: optional string grantor;            // Needed for grant
  6: optional PrincipalType grantorType; // Needed for grant
  7: optional bool grantOption;
}

struct GrantRevokeRoleResponse {
  1: optional bool success;
}

// namespace for tables
struct Database {
  1: string name,
  2: string description,
  3: string locationUri,
  4: map<string, string> parameters, // properties associated with the database
  5: optional PrincipalPrivilegeSet privileges,
  6: optional string ownerName,
  7: optional PrincipalType ownerType
```

```
}

// This object holds the information needed by SerDes
struct SerDeInfo {
  1: string name,                   // name of the serde, table name by default
  2: string serializationLib,       // usually the class that implements the extractor &
loader
  3: map<string, string> parameters // initialization parameters
}

// sort order of a column (column name along with asc(1)/desc(0))
struct Order {
  1: string col,      // sort column name
  2: i32    order     // asc(1) or desc(0)
}

// this object holds all the information about skewed table
struct SkewedInfo {
  1: list<string> skewedColNames, // skewed column names
  2: list<list<string>> skewedColValues, //skewed values
  3: map<list<string>, string> skewedColValueLocationMaps, //skewed value to location
mappings
}

// this object holds all the information about physical storage of the data belonging to a
table
struct StorageDescriptor {
  1: list<FieldSchema> cols,  // required (refer to types defined above)
  2: string location,         // defaults to <warehouse loc>/<db loc>/tablename
  3: string inputFormat,      // SequenceFileInputFormat (binary) or TextInputFormat`  or
custom format
  4: string outputFormat,     // SequenceFileOutputFormat (binary) or
IgnoreKeyTextOutputFormat or custom format
  5: bool   compressed,       // compressed or not
  6: i32    numBuckets,       // this must be specified if there are any dimension columns
  7: SerDeInfo    serdeInfo,  // serialization and deserialization information
  8: list<string> bucketCols, // reducer grouping columns and clustering columns and
bucketing columns`
  9: list<Order>  sortCols,   // sort order of the data in each bucket
  10: map<string, string> parameters, // any user supplied key value hash
  11: optional SkewedInfo skewedInfo, // skewed information
  12: optional bool   storedAsSubDirectories       // stored as subdirectories or not
}

// table information
struct Table {
  1: string tableName,                // name of the table
  2: string dbName,                   // database name ('default')
  3: string owner,                    // owner of this table
  4: i32    createTime,               // creation time of the table
  5: i32    lastAccessTime,           // last access time (usually this will be filled from
HDFS and shouldn't be relied on)
  6: i32    retention,                // retention time
  7: StorageDescriptor sd,            // storage descriptor of the table
  8: list<FieldSchema> partitionKeys, // partition keys of the table. only primitive types
are supported
  9: map<string, string> parameters,   // to store comments or any other user level
parameters
  10: string viewOriginalText,         // original view text, null for non-view
  11: string viewExpandedText,         // expanded view text, null for non-view
  12: string tableType,                // table type enum, e.g. EXTERNAL_TABLE
  13: optional PrincipalPrivilegeSet privileges,
  14: optional bool temporary=false,
  15: optional bool rewriteEnabled     // rewrite enabled or not
}

struct Partition {
  1: list<string> values, // string value is converted to appropriate partition key type
  2: string       dbName,
```

```
  3: string       tableName,
  4: i32          createTime,
  5: i32          lastAccessTime,
  6: StorageDescriptor   sd,
  7: map<string, string> parameters,
  8: optional PrincipalPrivilegeSet privileges
}

struct PartitionWithoutSD {
  1: list<string> values, // string value is converted to appropriate partition key type
  2: i32          createTime,
  3: i32          lastAccessTime,
  4: string       relativePath,
  5: map<string, string> parameters,
  6: optional PrincipalPrivilegeSet privileges
}

struct PartitionSpecWithSharedSD {
  1: list<PartitionWithoutSD> partitions,
  2: StorageDescriptor sd,
}

struct PartitionListComposingSpec {
  1: list<Partition> partitions
}

struct PartitionSpec {
  1: string dbName,
  2: string tableName,
  3: string rootPath,
  4: optional PartitionSpecWithSharedSD sharedSDPartitionSpec,
  5: optional PartitionListComposingSpec partitionList
}

struct Index {
  1: string       indexName, // unique with in the whole database namespace
  2: string       indexHandlerClass, // reserved
  3: string       dbName,
  4: string       origTableName,
  5: i32          createTime,
  6: i32          lastAccessTime,
  7: string       indexTableName,
  8: StorageDescriptor   sd,
  9: map<string, string> parameters,
  10: bool         deferredRebuild
}

// column statistics
struct BooleanColumnStatsData {
1: required i64 numTrues,
2: required i64 numFalses,
3: required i64 numNulls,
4: optional string bitVectors
}

struct DoubleColumnStatsData {
1: optional double lowValue,
2: optional double highValue,
3: required i64 numNulls,
4: required i64 numDVs,
5: optional string bitVectors
}

struct LongColumnStatsData {
1: optional i64 lowValue,
2: optional i64 highValue,
3: required i64 numNulls,
4: required i64 numDVs,
5: optional string bitVectors
```

```
}

struct StringColumnStatsData {
1: required i64 maxColLen,
2: required double avgColLen,
3: required i64 numNulls,
4: required i64 numDVs,
5: optional string bitVectors
}

struct BinaryColumnStatsData {
1: required i64 maxColLen,
2: required double avgColLen,
3: required i64 numNulls,
4: optional string bitVectors
}


struct Decimal {
1: required binary unscaled,
3: required i16 scale
}

struct DecimalColumnStatsData {
1: optional Decimal lowValue,
2: optional Decimal highValue,
3: required i64 numNulls,
4: required i64 numDVs,
5: optional string bitVectors
}

struct Date {
1: required i64 daysSinceEpoch
}

struct DateColumnStatsData {
1: optional Date lowValue,
2: optional Date highValue,
3: required i64 numNulls,
4: required i64 numDVs,
5: optional string bitVectors
}

union ColumnStatisticsData {
1: BooleanColumnStatsData booleanStats,
2: LongColumnStatsData longStats,
3: DoubleColumnStatsData doubleStats,
4: StringColumnStatsData stringStats,
5: BinaryColumnStatsData binaryStats,
6: DecimalColumnStatsData decimalStats,
7: DateColumnStatsData dateStats
}

struct ColumnStatisticsObj {
1: required string colName,
2: required string colType,
3: required ColumnStatisticsData statsData
}

struct ColumnStatisticsDesc {
1: required bool isTblLevel,
2: required string dbName,
3: required string tableName,
4: optional string partName,
5: optional i64 lastAnalyzed
}

struct ColumnStatistics {
1: required ColumnStatisticsDesc statsDesc,
```

```
2: required list<ColumnStatisticsObj> statsObj;
}

struct AggrStats {
1: required list<ColumnStatisticsObj> colStats,
2: required i64 partsFound // number of partitions for which stats were found
}

struct SetPartitionsStatsRequest {
1: required list<ColumnStatistics> colStats,
2: optional bool needMerge //stats need to be merged with the existing stats
}

// schema of the table/query results etc.
struct Schema {
 // column names, types, comments
 1: list<FieldSchema> fieldSchemas,  // delimiters etc
 2: map<string, string> properties
}

// Key-value store to be used with selected
// Metastore APIs (create, alter methods).
// The client can pass environment properties / configs that can be
// accessed in hooks.
struct EnvironmentContext {
  1: map<string, string> properties
}

struct PrimaryKeysRequest {
  1: required string db_name,
  2: required string tbl name
}

struct PrimaryKeysResponse {
  1: required list<SQLPrimaryKey> primaryKeys
}

struct ForeignKeysRequest {
  1: string parent_db_name,
  2: string parent tbl name,
  3: string foreign_db_name,
  4: string foreign tbl name
}

struct ForeignKeysResponse {
  1: required list<SQLForeignKey> foreignKeys
}

struct DropConstraintRequest {
  1: required string dbname,
  2: required string tablename,
  3: required string constraintname
}

struct AddPrimaryKeyRequest {
  1: required list<SQLPrimaryKey> primaryKeyCols
}

struct AddForeignKeyRequest {
  1: required list<SQLForeignKey> foreignKeyCols
}

// Return type for get partitions by expr
struct PartitionsByExprResult {
  1: required list<Partition> partitions,
  // Whether the results has any (currently, all) partitions which may or may not match
  2: required bool hasUnknownPartitions
}
```

```
struct PartitionsByExprRequest {
  1: required string dbName,
  2: required string tblName,
  3: required binary expr,
  4: optional string defaultPartitionName,
  5: optional i16 maxParts=-1
}

struct TableStatsResult {
  1: required list<ColumnStatisticsObj> tableStats
}

struct PartitionsStatsResult {
  1: required map<string, list<ColumnStatisticsObj>> partStats
}

struct TableStatsRequest {
 1: required string dbName,
 2: required string tblName,
 3: required list<string> colNames
}

struct PartitionsStatsRequest {
 1: required string dbName,
 2: required string tblName,
 3: required list<string> colNames,
 4: required list<string> partNames
}

// Return type for add_partitions_req
struct AddPartitionsResult {
  1: optional list<Partition> partitions,
}

// Request type for add_partitions_req
struct AddPartitionsRequest {
  1: required string dbName,
  2: required string tblName,
  3: required list<Partition> parts,
  4: required bool ifNotExists,
  5: optional bool needResult=true
}

// Return type for drop_partitions_req
struct DropPartitionsResult {
  1: optional list<Partition> partitions,
}

struct DropPartitionsExpr {
  1: required binary expr;
  2: optional i32 partArchiveLevel;
}

union RequestPartsSpec {
  1: list<string> names;
  2: list<DropPartitionsExpr> exprs;
}

// Request type for drop partitions req
// TODO: we might want to add "bestEffort" flag; where a subset can fail
struct DropPartitionsRequest {
  1: required string dbName,
  2: required string tblName,
  3: required RequestPartsSpec parts,
  4: optional bool deleteData,
  5: optional bool ifExists=true, // currently verified on client
  6: optional bool ignoreProtection,
  7: optional EnvironmentContext environmentContext,
  8: optional bool needResult=true
```

```
}

enum FunctionType {
  JAVA = 1,
}

enum ResourceType {
  JAR     = 1,
  FILE    = 2,
  ARCHIVE = 3,
}

struct ResourceUri {
  1: ResourceType resourceType,
  2: string       uri,
}

// User-defined function
struct Function {
  1: string           functionName,
  2: string           dbName,
  3: string           className,
  4: string           ownerName,
  5: PrincipalType    ownerType,
  6: i32              createTime,
  7: FunctionType     functionType,
  8: list<ResourceUri> resourceUris,
}

// Structs for transaction and locks
struct TxnInfo {
    1: required i64 id,
    2: required TxnState state,
    3: required string user,        // used in 'show transactions' to help admins find who
has open transactions
    4: required string hostname,    // used in 'show transactions' to help admins find who
has open transactions
    5: optional string agentInfo = "Unknown",
    6: optional i32 heartbeatCount=0,
    7: optional string metaInfo,
    8: optional i64 startedTime,
    9: optional i64 lastHeartbeatTime,
}

struct GetOpenTxnsInfoResponse {
    1: required i64 txn high water mark,
    2: required list<TxnInfo> open_txns,
}

struct GetOpenTxnsResponse {
    1: required i64 txn_high_water_mark,
    2: required set<i64> open txns,
    3: optional i64 min_open_txn, //since 1.3,2.2
}

struct OpenTxnRequest {
    1: required i32 num txns,
    2: required string user,
    3: required string hostname,
    4: optional string agentInfo = "Unknown",
}

struct OpenTxnsResponse {
    1: required list<i64> txn_ids,
}

struct AbortTxnRequest {
    1: required i64 txnid,
}
```

```
struct AbortTxnsRequest {
    1: required list<i64> txn_ids,
}

struct CommitTxnRequest {
    1: required i64 txnid,
}

struct LockComponent {
    1: required LockType type,
    2: required LockLevel level,
    3: required string dbname,
    4: optional string tablename,
    5: optional string partitionname,
    6: optional DataOperationType operationType = DataOperationType.UNSET,
    7: optional bool isAcid = false,
    8: optional bool isDynamicPartitionWrite = false
}

struct LockRequest {
    1: required list<LockComponent> component,
    2: optional i64 txnid,
    3: required string user,     // used in 'show locks' to help admins find who has open
locks
    4: required string hostname, // used in 'show locks' to help admins find who has open
locks
    5: optional string agentInfo = "Unknown",
}

struct LockResponse {
    1: required i64 lockid,
    2: required LockState state,
}

struct CheckLockRequest {
    1: required i64 lockid,
    2: optional i64 txnid,
    3: optional i64 elapsed_ms,
}

struct UnlockRequest {
    1: required i64 lockid,
}

struct ShowLocksRequest {
    1: optional string dbname,
    2: optional string tablename,
    3: optional string partname,
    4: optional bool isExtended=false,
}

struct ShowLocksResponseElement {
    1: required i64 lockid,
    2: required string dbname,
    3: optional string tablename,
    4: optional string partname,
    5: required LockState state,
    6: required LockType type,
    7: optional i64 txnid,
    8: required i64 lastheartbeat,
    9: optional i64 acquiredat,
    10: required string user,
    11: required string hostname,
    12: optional i32 heartbeatCount = 0,
    13: optional string agentInfo,
    14: optional i64 blockedByExtId,
    15: optional i64 blockedByIntId,
    16: optional i64 lockIdInternal,
```

```
}

struct ShowLocksResponse {
    1: list<ShowLocksResponseElement> locks,
}

struct HeartbeatRequest {
    1: optional i64 lockid,
    2: optional i64 txnid
}

struct HeartbeatTxnRangeRequest {
    1: required i64 min,
    2: required i64 max
}

struct HeartbeatTxnRangeResponse {
    1: required set<i64> aborted,
    2: required set<i64> nosuch
}

struct CompactionRequest {
    1: required string dbname,
    2: required string tablename,
    3: optional string partitionname,
    4: required CompactionType type,
    5: optional string runas,
    6: optional map<string, string> properties
}

struct CompactionResponse {
    1: required i64 id,
    2: required string state,
    3: required bool accepted
}

struct ShowCompactRequest {
}

struct ShowCompactResponseElement {
    1: required string dbname,
    2: required string tablename,
    3: optional string partitionname,
    4: required CompactionType type,
    5: required string state,
    6: optional string workerid,
    7: optional i64 start,
    8: optional string runAs,
    9: optional i64 hightestTxnId, // Highest Txn ID handled by this compaction
    10: optional string metaInfo,
    11: optional i64 endTime,
    12: optional string hadoopJobId = "None",
    13: optional i64 id,
}

struct ShowCompactResponse {
    1: required list<ShowCompactResponseElement> compacts,
}

struct AddDynamicPartitions {
    1: required i64 txnid,
    2: required string dbname,
    3: required string tablename,
    4: required list<string> partitionnames,
    5: optional DataOperationType operationType = DataOperationType.UNSET
}

struct NotificationEventRequest {
    1: required i64 lastEvent,
```

```
    2: optional i32 maxEvents,
}

struct NotificationEvent {
    1: required i64 eventId,
    2: required i32 eventTime,
    3: required string eventType,
    4: optional string dbName,
    5: optional string tableName,
    6: required string message,
    7: optional string messageFormat,
}

struct NotificationEventResponse {
    1: required list<NotificationEvent> events,
}

struct CurrentNotificationEventId {
    1: required i64 eventId,
}

struct InsertEventRequestData {
    1: required list<string> filesAdded,
    // Checksum of files (hex string of checksum byte payload)
    2: optional list<string> filesAddedChecksum,
}

union FireEventRequestData {
    1: InsertEventRequestData insertData
}

struct FireEventRequest {
    1: required bool successful,
    2: required FireEventRequestData data
    // dbname, tablename, and partition vals are included as optional in the top level event
rather than placed in each type of
    // subevent as I assume they'll be used across most event types.
    3: optional string dbName,
    4: optional string tableName,
    5: optional list<string> partitionVals,
}

struct FireEventResponse {
    // NOP for now, this is just a place holder for future responses
}

struct MetadataPpdResult {
  1: optional binary metadata,
  2: optional binary includeBitset
}

// Return type for get file metadata by expr
struct GetFileMetadataByExprResult {
  1: required map<i64, MetadataPpdResult> metadata,
  2: required bool isSupported
}

enum FileMetadataExprType {
  ORC SARG = 1
}


// Request type for get file metadata by expr
struct GetFileMetadataByExprRequest {
  1: required list<i64> fileIds,
  2: required binary expr,
  3: optional bool doGetFooters,
  4: optional FileMetadataExprType type
}
```

```
// Return type for get_file_metadata
struct GetFileMetadataResult {
  1: required map<i64, binary> metadata,
  2: required bool isSupported
}

// Request type for get_file_metadata
struct GetFileMetadataRequest {
  1: required list<i64> fileIds
}

// Return type for put_file_metadata
struct PutFileMetadataResult {
}

// Request type for put file metadata
struct PutFileMetadataRequest {
  1: required list<i64> fileIds,
  2: required list<binary> metadata,
  3: optional FileMetadataExprType type
}

// Return type for clear_file_metadata
struct ClearFileMetadataResult {
}

// Request type for clear_file_metadata
struct ClearFileMetadataRequest {
  1: required list<i64> fileIds
}

// Return type for cache_file_metadata
struct CacheFileMetadataResult {
  1: required bool isSupported
}

// Request type for cache_file_metadata
struct CacheFileMetadataRequest {
  1: required string dbName,
  2: required string tblName,
  3: optional string partName,
  4: optional bool isAllParts
}

struct GetAllFunctionsResponse {
  1: optional list<Function> functions
}

enum ClientCapability {
  TEST_CAPABILITY = 1
}


struct ClientCapabilities {
  1: required list<ClientCapability> values
}

struct GetTableRequest {
  1: required string dbName,
  2: required string tblName,
  3: optional ClientCapabilities capabilities
}

struct GetTableResult {
  1: required Table table
}

struct GetTablesRequest {
```

```
  1: required string dbName,
  2: optional list<string> tblNames,
  3: optional ClientCapabilities capabilities
}

struct GetTablesResult {
  1: required list<Table> tables
}

struct TableMeta {
  1: required string dbName;
  2: required string tableName;
  3: required string tableType;
  4: optional string comments;
}

exception MetaException {
  1: string message
}

exception UnknownTableException {
  1: string message
}

exception UnknownDBException {
  1: string message
}

exception AlreadyExistsException {
  1: string message
}

exception InvalidPartitionException {
  1: string message
}

exception UnknownPartitionException {
  1: string message
}

exception InvalidObjectException {
  1: string message
}

exception NoSuchObjectException {
  1: string message
}

exception IndexAlreadyExistsException {
  1: string message
}

exception InvalidOperationException {
  1: string message
}

exception ConfigValSecurityException {
  1: string message
}

exception InvalidInputException {
  1: string message
}

// Transaction and lock exceptions
exception NoSuchTxnException {
    1: string message
}
```

```
exception TxnAbortedException {
    1: string message
}

exception TxnOpenException {
    1: string message
}

exception NoSuchLockException {
    1: string message
}

/**
* This interface is live.
*/
service ThriftHiveMetastore extends fb303.FacebookService
{
  string getMetaConf(1:string key) throws(1:MetaException o1)
  void setMetaConf(1:string key, 2:string value) throws(1:MetaException o1)

  void create_database(1:Database database) throws(1:AlreadyExistsException o1,
2:InvalidObjectException o2, 3:MetaException o3)
  Database get_database(1:string name) throws(1:NoSuchObjectException o1, 2:MetaException o2)
  void drop_database(1:string name, 2:bool deleteData, 3:bool cascade)
throws(1:NoSuchObjectException o1, 2:InvalidOperationException o2, 3:MetaException o3)
  list<string> get_databases(1:string pattern) throws(1:MetaException o1)
  list<string> get_all_databases() throws(1:MetaException o1)
  void alter_database(1:string dbname, 2:Database db) throws(1:MetaException o1,
2:NoSuchObjectException o2)

  // returns the type with given name (make seperate calls for the dependent types if needed)
  Type get_type(1:string name)  throws(1:MetaException o1, 2:NoSuchObjectException o2)
  bool create_type(1:Type type) throws(1:AlreadyExistsException o1, 2:InvalidObjectException
o2, 3:MetaException o3)
  bool drop_type(1:string type) throws(1:MetaException o1, 2:NoSuchObjectException o2)
  map<string, Type> get type all(1:string name)
                                throws(1:MetaException o2)

  // Gets a list of FieldSchemas describing the columns of a particular table
  list<FieldSchema> get fields(1: string db name, 2: string table name) throws (1:
MetaException o1, 2: UnknownTableException o2, 3: UnknownDBException o3),
  list<FieldSchema> get fields with environment context(1: string db name, 2: string
table_name, 3:EnvironmentContext environment_context) throws (1: MetaException o1, 2:
UnknownTableException o2, 3: UnknownDBException o3)

  // Gets a list of FieldSchemas describing both the columns and the partition keys of a
particular table
  list<FieldSchema> get schema(1: string db name, 2: string table name) throws (1:
MetaException o1, 2: UnknownTableException o2, 3: UnknownDBException o3)
  list<FieldSchema> get_schema_with_environment_context(1: string db_name, 2: string
table_name, 3:EnvironmentContext environment_context) throws (1: MetaException o1, 2:
UnknownTableException o2, 3: UnknownDBException o3)

  // create a Hive table. Following fields must be set
  // tableName
  // database        (only 'default' for now until Hive QL supports databases)
  // owner           (not needed, but good to have for tracking purposes)
  // sd.cols         (list of field schemas)
  // sd.inputFormat  (SequenceFileInputFormat (binary like falcon tables or u full) or
TextInputFormat)
  // sd.outputFormat (SequenceFileInputFormat (binary) or TextInputFormat)
  // sd.serdeInfo.serializationLib (SerDe class name eg
org.apache.hadoop.hive.serde.simple meta.MetadataTypedColumnsetSerDe
  // * See notes on DDL_TIME
  void create table(1:Table tbl) throws(1:AlreadyExistsException o1, 2:InvalidObjectException
o2, 3:MetaException o3, 4:NoSuchObjectException o4)
  void create_table_with_environment_context(1:Table tbl,
      2:EnvironmentContext environment_context)
      throws (1:AlreadyExistsException o1,
```

```
              2:InvalidObjectException o2, 3:MetaException o3,
              4:NoSuchObjectException o4)
  void create_table_with_constraints(1:Table tbl, 2: list<SQLPrimaryKey> primaryKeys, 3:
list<SQLForeignKey> foreignKeys)
      throws (1:AlreadyExistsException o1,
              2:InvalidObjectException o2, 3:MetaException o3,
              4:NoSuchObjectException o4)
  void drop_constraint(1:DropConstraintRequest req)
      throws(1:NoSuchObjectException o1, 2:MetaException o3)
  void add_primary_key(1:AddPrimaryKeyRequest req)
      throws(1:NoSuchObjectException o1, 2:MetaException o2)
  void add_foreign_key(1:AddForeignKeyRequest req)
      throws(1:NoSuchObjectException o1, 2:MetaException o2)

  // drops the table and all the partitions associated with it if the table has partitions
  // delete data (including partitions) if deleteData is set to true
  void drop_table(1:string dbname, 2:string name, 3:bool deleteData)
                       throws(1:NoSuchObjectException o1, 2:MetaException o3)
  void drop_table_with_environment_context(1:string dbname, 2:string name, 3:bool deleteData,
      4:EnvironmentContext environment_context)
                       throws(1:NoSuchObjectException o1, 2:MetaException o3)
  list<string> get_tables(1: string db_name, 2: string pattern) throws (1: MetaException o1)
  list<string> get_tables_by_type(1: string db_name, 2: string pattern, 3: string tableType)
throws (1: MetaException o1)
  list<TableMeta> get_table_meta(1: string db_patterns, 2: string tbl_patterns, 3:
list<string> tbl_types)
                       throws (1: MetaException o1)
  list<string> get_all_tables(1: string db_name) throws (1: MetaException o1)

  Table get_table(1:string dbname, 2:string tbl_name)
                       throws (1:MetaException o1, 2:NoSuchObjectException o2)
  list<Table> get_table_objects_by_name(1:string dbname, 2:list<string> tbl_names)
  GetTableResult get_table_req(1:GetTableRequest req)
                       throws (1:MetaException o1, 2:NoSuchObjectException o2)
  GetTablesResult get_table_objects_by_name_req(1:GetTablesRequest req)


                                    throws (1:MetaException o1, 2:InvalidOperationException
o2, 3:UnknownDBException o3)

  // Get a list of table names that match a filter.
  // The filter operators are LIKE, <, <=, >, >=, =, <>
  //
  // In the filter statement, values interpreted as strings must be enclosed in quotes,
  // while values interpreted as integers should not be.  Strings and integers are the only
  // supported value types.
  //
  // The currently supported key names in the filter are:
  // Constants.HIVE_FILTER_FIELD_OWNER, which filters on the tables' owner's name
  //   and supports all filter operators
  // Constants.HIVE_FILTER_FIELD_LAST_ACCESS, which filters on the last access times
  //   and supports all filter operators except LIKE
  // Constants.HIVE_FILTER_FIELD_PARAMS, which filters on the tables' parameter keys and
values
  //   and only supports the filter operators = and <>.
  //   Append the parameter key name to HIVE_FILTER_FIELD_PARAMS in the filter statement.
  //   For example, to filter on parameter keys called "retention", the key name in the
filter
  //   statement should be Constants.HIVE_FILTER_FIELD_PARAMS + "retention"
  //   Also, = and <> only work for keys that exist
  //   in the tables. E.g., if you are looking for tables where key1 <> value, it will only
  //   look at tables that have a value for the parameter key1.
  // Some example filter statements include:
  // filter = Constants.HIVE_FILTER_FIELD_OWNER + " like \".*test.*\" and " +
  //   Constants.HIVE_FILTER_FIELD_LAST_ACCESS + " = 0";
  // filter = Constants.HIVE_FILTER_FIELD_PARAMS + "retention = \"30\" or " +
  //   Constants.HIVE_FILTER_FIELD_PARAMS + "retention = \"90\""
  // @param dbName
  //          The name of the database from which you will retrieve the table names
```

```
  // @param filterType
  //          The type of filter
  // @param filter
  //          The filter string
  // @param max_tables
  //          The maximum number of tables returned
  // @return  A list of table names that match the desired filter
  list<string> get_table_names_by_filter(1:string dbname, 2:string filter, 3:i16 max_tables=-
1)
                       throws (1:MetaException o1, 2:InvalidOperationException o2,
3:UnknownDBException o3)

  // alter table applies to only future partitions not for existing partitions
  // * See notes on DDL_TIME
  void alter_table(1:string dbname, 2:string tbl_name, 3:Table new_tbl)
                       throws (1:InvalidOperationException o1, 2:MetaException o2)
  void alter table with environment context(1:string dbname, 2:string tbl name,
      3:Table new_tbl, 4:EnvironmentContext environment_context)
      throws (1:InvalidOperationException o1, 2:MetaException o2)
  // alter table not only applies to future partitions but also cascade to existing
partitions
  void alter_table_with_cascade(1:string dbname, 2:string tbl_name, 3:Table new_tbl, 4:bool
cascade)
                       throws (1:InvalidOperationException o1, 2:MetaException o2)
  // the following applies to only tables that have partitions
  // * See notes on DDL_TIME
  Partition add_partition(1:Partition new_part)
                       throws(1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  Partition add_partition_with_environment_context(1:Partition new_part,
      2:EnvironmentContext environment context)
      throws (1:InvalidObjectException o1, 2:AlreadyExistsException o2,
      3:MetaException o3)
  i32 add partitions(1:list<Partition> new parts)
                       throws(1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  i32 add_partitions_pspec(1:list<PartitionSpec> new_parts)
                       throws(1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  Partition append partition(1:string db name, 2:string tbl name, 3:list<string> part vals)
                       throws (1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  AddPartitionsResult add_partitions_req(1:AddPartitionsRequest request)
                       throws(1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  Partition append partition with environment context(1:string db name, 2:string tbl name,
      3:list<string> part_vals, 4:EnvironmentContext environment_context)
                       throws (1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  Partition append_partition_by_name(1:string db_name, 2:string tbl_name, 3:string part_name)
                       throws (1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  Partition append_partition_by_name_with_environment_context(1:string db_name, 2:string
tbl name,
      3:string part_name, 4:EnvironmentContext environment_context)
                       throws (1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  bool drop partition(1:string db name, 2:string tbl name, 3:list<string> part vals, 4:bool
deleteData)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)
  bool drop_partition_with_environment_context(1:string db_name, 2:string tbl_name,
      3:list<string> part_vals, 4:bool deleteData, 5:EnvironmentContext environment_context)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)
  bool drop_partition_by_name(1:string db_name, 2:string tbl_name, 3:string part_name, 4:bool
deleteData)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)
  bool drop_partition_by_name_with_environment_context(1:string db_name, 2:string tbl_name,
      3:string part_name, 4:bool deleteData, 5:EnvironmentContext environment_context)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)
```

```
  DropPartitionsResult drop_partitions_req(1: DropPartitionsRequest req)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)

  Partition get_partition(1:string db_name, 2:string tbl_name, 3:list<string> part_vals)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)
  Partition exchange partition(1:map<string, string> partitionSpecs, 2:string source db,
      3:string source_table_name, 4:string dest_db, 5:string dest_table_name)
      throws(1:MetaException o1, 2:NoSuchObjectException o2, 3:InvalidObjectException o3,
      4:InvalidInputException o4)

  list<Partition> exchange partitions(1:map<string, string> partitionSpecs, 2:string
source_db,
      3:string source_table_name, 4:string dest_db, 5:string dest_table_name)
      throws(1:MetaException o1, 2:NoSuchObjectException o2, 3:InvalidObjectException o3,
      4:InvalidInputException o4)

  Partition get_partition_with_auth(1:string db_name, 2:string tbl_name, 3:list<string>
part_vals,
      4: string user_name, 5: list<string> group_names) throws(1:MetaException o1,
2:NoSuchObjectException o2)

  Partition get_partition_by_name(1:string db_name 2:string tbl_name, 3:string part_name)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // returns all the partitions for this table in reverse chronological order.
  // If max parts is given then it will return only that many.
  list<Partition> get_partitions(1:string db_name, 2:string tbl_name, 3:i16 max_parts=-1)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)
  list<Partition> get_partitions_with_auth(1:string db_name, 2:string tbl_name, 3:i16
max_parts=-1,
     4: string user_name, 5: list<string> group_names) throws(1:NoSuchObjectException o1,
2:MetaException o2)

  list<PartitionSpec> get_partitions_pspec(1:string db_name, 2:string tbl_name, 3:i32
max_parts=-1)
                       throws(1:NoSuchObjectException o1, 2:MetaException o2)

  list<string> get_partition_names(1:string db_name, 2:string tbl_name, 3:i16 max_parts=-1)
                       throws(1:MetaException o2)

  // get_partition*_ps methods allow filtering by a partial partition specification,
  // as needed for dynamic partitions. The values that are not restricted should
  // be empty strings. Nulls were considered (instead of "") but caused errors in
  // generated Python code. The size of part_vals may be smaller than the
  // number of partition columns - the unspecified values are considered the same
  // as "".
  list<Partition> get_partitions_ps(1:string db_name 2:string tbl_name
       3:list<string> part_vals, 4:i16 max_parts=-1)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)
  list<Partition> get_partitions_ps_with_auth(1:string db_name, 2:string tbl_name,
3:list<string> part_vals, 4:i16 max_parts=-1,
     5: string user_name, 6: list<string> group_names) throws(1:NoSuchObjectException o1,
2:MetaException o2)

  list<string> get_partition_names_ps(1:string db_name,
       2:string tbl_name, 3:list<string> part_vals, 4:i16 max_parts=-1)
                          throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // get the partitions matching the given partition filter
  list<Partition> get_partitions_by_filter(1:string db_name 2:string tbl_name
    3:string filter, 4:i16 max_parts=-1)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // List partitions as PartitionSpec instances.
  list<PartitionSpec> get_part_specs_by_filter(1:string db_name 2:string tbl_name
    3:string filter, 4:i32 max_parts=-1)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // get the partitions matching the given partition filter
```

```
  // unlike get_partitions_by_filter, takes serialized hive expression, and with that can
work
  // with any filter (get_partitions_by_filter only works if the filter can be pushed down to
JDOQL.
  PartitionsByExprResult get_partitions_by_expr(1:PartitionsByExprRequest req)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // get the partitions matching the given partition filter
  i32 get_num_partitions_by_filter(1:string db_name 2:string tbl_name 3:string filter)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // get partitions give a list of partition names
  list<Partition> get_partitions_by_names(1:string db_name 2:string tbl_name 3:list<string>
names)
                       throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // changes the partition to the new partition object. partition is identified from the part
values
  // in the new part
  // * See notes on DDL_TIME
  void alter_partition(1:string db_name, 2:string tbl_name, 3:Partition new_part)
                       throws (1:InvalidOperationException o1, 2:MetaException o2)

  // change a list of partitions. All partitions are altered atomically and all
  // prehooks are fired together followed by all post hooks
  void alter_partitions(1:string db_name, 2:string tbl_name, 3:list<Partition> new_parts)
                       throws (1:InvalidOperationException o1, 2:MetaException o2)
  void alter_partitions_with_environment_context(1:string db_name, 2:string tbl_name,
3:list<Partition> new parts, 4:EnvironmentContext environment context) throws
(1:InvalidOperationException o1, 2:MetaException o2)

  void alter_partition_with_environment_context(1:string db_name,
      2:string tbl_name, 3:Partition new_part,
      4:EnvironmentContext environment context)
      throws (1:InvalidOperationException o1, 2:MetaException o2)

  // rename the old partition to the new partition object by changing old part values to the
part values
  // in the new_part. old partition is identified from part_vals.
  // partition keys in new part should be the same as those in old partition.
  void rename_partition(1:string db_name, 2:string tbl_name, 3:list<string> part_vals,
4:Partition new part)
                       throws (1:InvalidOperationException o1, 2:MetaException o2)

  // returns whether or not the partition name is valid based on the value of the config
  // hive.metastore.partition.name.whitelist.pattern
  bool partition_name_has_valid_characters(1:list<string> part_vals, 2:bool throw_exception)
       throws(1: MetaException o1)

  // gets the value of the configuration key in the metastore server. returns
  // defaultValue if the key does not exist. if the configuration key does not
  // begin with "hive", "mapred", or "hdfs", a ConfigValSecurityException is
  // thrown.
  string get config value(1:string name, 2:string defaultValue)
                         throws(1:ConfigValSecurityException o1)

  // converts a partition name into a partition values array
  list<string> partition name to vals(1: string part name)
                         throws(1: MetaException o1)
  // converts a partition name into a partition specification (a mapping from
  // the partition cols to the values)
  map<string, string> partition_name_to_spec(1: string part_name)
                         throws(1: MetaException o1)

  void markPartitionForEvent(1:string db name, 2:string tbl name, 3:map<string,string>
part_vals,
                  4:PartitionEventType eventType) throws (1: MetaException o1, 2:
NoSuchObjectException o2,
```

```
                    3: UnknownDBException o3, 4: UnknownTableException o4, 5:
UnknownPartitionException o5,
                    6: InvalidPartitionException o6)
  bool isPartitionMarkedForEvent(1:string db_name, 2:string tbl_name, 3:map<string,string>
part_vals,
                    4: PartitionEventType eventType) throws (1: MetaException o1,
2:NoSuchObjectException o2,
                    3: UnknownDBException o3, 4: UnknownTableException o4, 5:
UnknownPartitionException o5,
                    6: InvalidPartitionException o6)

  //index
  Index add_index(1:Index new_index, 2: Table index_table)
                        throws(1:InvalidObjectException o1, 2:AlreadyExistsException o2,
3:MetaException o3)
  void alter_index(1:string dbname, 2:string base_tbl_name, 3:string idx_name, 4:Index
new idx)
                        throws (1:InvalidOperationException o1, 2:MetaException o2)
  bool drop index by name(1:string db name, 2:string tbl name, 3:string index name, 4:bool
deleteData)
                        throws(1:NoSuchObjectException o1, 2:MetaException o2)
  Index get_index_by_name(1:string db_name 2:string tbl_name, 3:string index_name)
                        throws(1:MetaException o1, 2:NoSuchObjectException o2)

  list<Index> get_indexes(1:string db_name, 2:string tbl_name, 3:i16 max_indexes=-1)
                        throws(1:NoSuchObjectException o1, 2:MetaException o2)
  list<string> get_index_names(1:string db_name, 2:string tbl_name, 3:i16 max_indexes=-1)
                        throws(1:MetaException o2)

 //primary keys and foreign keys
  PrimaryKeysResponse get primary keys(1:PrimaryKeysRequest request)
                        throws(1:MetaException o1, 2:NoSuchObjectException o2)
  ForeignKeysResponse get_foreign_keys(1:ForeignKeysRequest request)
                        throws(1:MetaException o1, 2:NoSuchObjectException o2)

  // column statistics interfaces

  // update APIs persist the column statistics object(s) that are passed in. If statistics
already
  // exists for one or more columns, the existing statistics will be overwritten. The update
APIs
  // validate that the dbName, tableName, partName, colName[] passed in as part of the
ColumnStatistics
  // struct are valid, throws InvalidInputException/NoSuchObjectException if found to be
invalid
  bool update table column statistics(1:ColumnStatistics stats obj) throws
(1:NoSuchObjectException o1,
              2:InvalidObjectException o2, 3:MetaException o3, 4:InvalidInputException o4)
  bool update_partition_column_statistics(1:ColumnStatistics stats_obj) throws
(1:NoSuchObjectException o1,
              2:InvalidObjectException o2, 3:MetaException o3, 4:InvalidInputException o4)

  // get APIs return the column statistics corresponding to db_name, tbl_name, [part_name],
col name if
  // such statistics exists. If the required statistics doesn't exist, get APIs throw
NoSuchObjectException
  // For instance, if get table column statistics is called on a partitioned table for which
only
  // partition level column stats exist, get table column statistics will throw
NoSuchObjectException
  ColumnStatistics get_table_column_statistics(1:string db_name, 2:string tbl_name, 3:string
col_name) throws
              (1:NoSuchObjectException o1, 2:MetaException o2, 3:InvalidInputException o3,
4:InvalidObjectException o4)
  ColumnStatistics get partition column statistics(1:string db name, 2:string tbl name,
3:string part_name,
               4:string col_name) throws (1:NoSuchObjectException o1, 2:MetaException o2,
               3:InvalidInputException o3, 4:InvalidObjectException o4)
  TableStatsResult get_table_statistics_req(1:TableStatsRequest request) throws
```

```
              (1:NoSuchObjectException o1, 2:MetaException o2)
  PartitionsStatsResult get_partitions_statistics_req(1:PartitionsStatsRequest request)
throws
              (1:NoSuchObjectException o1, 2:MetaException o2)
  AggrStats get_aggr_stats_for(1:PartitionsStatsRequest request) throws
              (1:NoSuchObjectException o1, 2:MetaException o2)
  bool set_aggr_stats_for(1:SetPartitionsStatsRequest request) throws
              (1:NoSuchObjectException o1, 2:InvalidObjectException o2, 3:MetaException o3,
4:InvalidInputException o4)


  // delete APIs attempt to delete column statistics, if found, associated with a given
db_name, tbl_name, [part_name]
  // and col_name. If the delete API doesn't find the statistics record in the metastore,
throws NoSuchObjectException
  // Delete API validates the input and if the input is invalid throws
InvalidInputException/InvalidObjectException.
  bool delete_partition_column_statistics(1:string db_name, 2:string tbl_name, 3:string
part name, 4:string col name) throws
              (1:NoSuchObjectException o1, 2:MetaException o2, 3:InvalidObjectException o3,
               4:InvalidInputException o4)
  bool delete_table_column_statistics(1:string db_name, 2:string tbl_name, 3:string col_name)
throws
              (1:NoSuchObjectException o1, 2:MetaException o2, 3:InvalidObjectException o3,
               4:InvalidInputException o4)

  //
  // user-defined functions
  //

  void create function(1:Function func)
      throws (1:AlreadyExistsException o1,
              2:InvalidObjectException o2,
              3:MetaException o3,
              4:NoSuchObjectException o4)

  void drop_function(1:string dbName, 2:string funcName)
      throws (1:NoSuchObjectException o1, 2:MetaException o3)

  void alter function(1:string dbName, 2:string funcName, 3:Function newFunc)
      throws (1:InvalidOperationException o1, 2:MetaException o2)

  list<string> get_functions(1:string dbName, 2:string pattern)
      throws (1:MetaException o1)
  Function get_function(1:string dbName, 2:string funcName)
      throws (1:MetaException o1, 2:NoSuchObjectException o2)

  GetAllFunctionsResponse get all functions() throws (1:MetaException o1)

  //authorization privileges

  bool create role(1:Role role) throws(1:MetaException o1)
  bool drop_role(1:string role_name) throws(1:MetaException o1)
  list<string> get role names() throws(1:MetaException o1)
  // Deprecated, use grant_revoke_role()
  bool grant_role(1:string role_name, 2:string principal_name, 3:PrincipalType
principal type,
    4:string grantor, 5:PrincipalType grantorType, 6:bool grant option)
throws(1:MetaException o1)
  // Deprecated, use grant_revoke_role()
  bool revoke_role(1:string role_name, 2:string principal_name, 3:PrincipalType
principal_type)
                        throws(1:MetaException o1)
  list<Role> list_roles(1:string principal_name, 2:PrincipalType principal_type)
throws(1:MetaException o1)
  GrantRevokeRoleResponse grant_revoke_role(1:GrantRevokeRoleRequest request)
throws(1:MetaException o1)

  // get all role-grants for users/roles that have been granted the given role
```

```
  // Note that in the returned list of RolePrincipalGrants, the roleName is
  // redundant as it would match the role_name argument of this function
  GetPrincipalsInRoleResponse get_principals_in_role(1: GetPrincipalsInRoleRequest request)
throws(1:MetaException o1)

  // get grant information of all roles granted to the given principal
  // Note that in the returned list of RolePrincipalGrants, the principal name,type is
  // redundant as it would match the principal name,type arguments of this function
  GetRoleGrantsForPrincipalResponse get_role_grants_for_principal(1:
GetRoleGrantsForPrincipalRequest request) throws(1:MetaException o1)

  PrincipalPrivilegeSet get_privilege_set(1:HiveObjectRef hiveObject, 2:string user_name,
    3: list<string> group_names) throws(1:MetaException o1)
  list<HiveObjectPrivilege> list_privileges(1:string principal_name, 2:PrincipalType
principal_type,
    3: HiveObjectRef hiveObject) throws(1:MetaException o1)

  // Deprecated, use grant_revoke_privileges()
  bool grant privileges(1:PrivilegeBag privileges) throws(1:MetaException o1)
  // Deprecated, use grant_revoke_privileges()
  bool revoke_privileges(1:PrivilegeBag privileges) throws(1:MetaException o1)
  GrantRevokePrivilegeResponse grant_revoke_privileges(1:GrantRevokePrivilegeRequest request)
throws(1:MetaException o1);

  // this is used by metastore client to send UGI information to metastore server immediately
  // after setting up a connection.
  list<string> set_ugi(1:string user_name, 2:list<string> group_names) throws
(1:MetaException o1)

  //Authentication (delegation token) interfaces

  // get metastore server delegation token for use from the map/reduce tasks to authenticate
  // to metastore server
  string get delegation token(1:string token owner, 2:string renewer kerberos principal name)
    throws (1:MetaException o1)

  // method to renew delegation token obtained from metastore server
  i64 renew_delegation_token(1:string token_str_form) throws (1:MetaException o1)

  // method to cancel delegation token obtained from metastore server
  void cancel_delegation_token(1:string token_str_form) throws (1:MetaException o1)

  // add a delegation token
  bool add_token(1:string token_identifier, 2:string delegation_token)

  // remove a delegation token
  bool remove_token(1:string token_identifier)

  // get a delegation token by identifier
  string get_token(1:string token_identifier)

  // get all delegation token identifiers
  list<string> get_all_token_identifiers()

  // add master key
  i32 add_master_key(1:string key) throws (1:MetaException o1)

  // update master key
  void update master key(1:i32 seq number, 2:string key) throws (1:NoSuchObjectException o1,
2:MetaException o2)

  // remove master key
  bool remove master key(1:i32 key seq)

  // get master keys
  list<string> get_master_keys()

  // Transaction and lock management calls
  // Get just list of open transactions
```

```
  GetOpenTxnsResponse get_open_txns()
  // Get list of open transactions with state (open, aborted)
  GetOpenTxnsInfoResponse get_open_txns_info()
  OpenTxnsResponse open_txns(1:OpenTxnRequest rqst)
  void abort_txn(1:AbortTxnRequest rqst) throws (1:NoSuchTxnException o1)
  void abort_txns(1:AbortTxnsRequest rqst) throws (1:NoSuchTxnException o1)
  void commit_txn(1:CommitTxnRequest rqst) throws (1:NoSuchTxnException o1,
2:TxnAbortedException o2)
  LockResponse lock(1:LockRequest rqst) throws (1:NoSuchTxnException o1,
2:TxnAbortedException o2)
  LockResponse check_lock(1:CheckLockRequest rqst)
    throws (1:NoSuchTxnException o1, 2:TxnAbortedException o2, 3:NoSuchLockException o3)
  void unlock(1:UnlockRequest rqst) throws (1:NoSuchLockException o1, 2:TxnOpenException o2)
  ShowLocksResponse show_locks(1:ShowLocksRequest rqst)
  void heartbeat(1:HeartbeatRequest ids) throws (1:NoSuchLockException o1,
2:NoSuchTxnException o2, 3:TxnAbortedException o3)
  HeartbeatTxnRangeResponse heartbeat_txn_range(1:HeartbeatTxnRangeRequest txns)
  void compact(1:CompactionRequest rqst)
  CompactionResponse compact2(1:CompactionRequest rqst)
  ShowCompactResponse show_compact(1:ShowCompactRequest rqst)
  void add_dynamic_partitions(1:AddDynamicPartitions rqst) throws (1:NoSuchTxnException o1,
2:TxnAbortedException o2)

  // Notification logging calls
  NotificationEventResponse get_next_notification(1:NotificationEventRequest rqst)
  CurrentNotificationEventId get_current_notificationEventId()
  FireEventResponse fire_listener_event(1:FireEventRequest rqst)
  void flushCache()

  GetFileMetadataByExprResult get_file_metadata_by_expr(1:GetFileMetadataByExprRequest req)
  GetFileMetadataResult get_file_metadata(1:GetFileMetadataRequest req)
  PutFileMetadataResult put_file_metadata(1:PutFileMetadataRequest req)
  ClearFileMetadataResult clear_file_metadata(1:ClearFileMetadataRequest req)
  CacheFileMetadataResult cache_file_metadata(1:CacheFileMetadataRequest req)

}

// * Note about the DDL_TIME: When creating or altering a table or a partition,
// if the DDL_TIME is not set, the current time will be used.

// For storing info about archived partitions in parameters

// Whether the partition is archived
const string IS_ARCHIVED = "is_archived",
// The original location of the partition, before archiving. After archiving,
// this directory will contain the archive. When the partition
// is dropped, this directory will be deleted
const string ORIGINAL_LOCATION = "original_location",

// Whether or not the table is considered immutable - immutable tables can only be
// overwritten or created if unpartitioned, or if partitioned, partitions inside them
// can only be overwritten or created. Immutability supports write-once and replace
// semantics, but not append.
const string IS_IMMUTABLE = "immutable",

// these should be needed only for backward compatibility with filestore
const string META_TABLE_COLUMNS   = "columns",
const string META_TABLE_COLUMN_TYPES   = "columns.types",
const string BUCKET_FIELD_NAME    = "bucket_field_name",
const string BUCKET_COUNT         = "bucket_count",
const string FIELD_TO_DIMENSION   = "field_to_dimension",
const string META_TABLE_NAME      = "name",
const string META_TABLE_DB        = "db",
const string META_TABLE_LOCATION  = "location",
const string META_TABLE_SERDE     = "serde",
const string META_TABLE_PARTITION_COLUMNS = "partition_columns",
const string META_TABLE_PARTITION_COLUMN_TYPES = "partition_columns.types",
const string FILE_INPUT_FORMAT    = "file.inputformat",
const string FILE_OUTPUT_FORMAT   = "file.outputformat",
```

```
const string META_TABLE_STORAGE   = "storage_handler",
const string TABLE_IS_TRANSACTIONAL = "transactional",
const string TABLE_NO_AUTO_COMPACT = "no_auto_compaction",
const string TABLE_TRANSACTIONAL_PROPERTIES = "transactional_properties",
```

6.2 Hive_metastore Thrift APIs

The **Apache Thrift** schema for the **Hive Metastore** can be used to generate a JavaScript API. This section provides the subset list of generated APIs that are supported by this protocol.

API	Signature
get_database	`Database get_database(1:string name) throws(1:NoSuchObjectException o1, 2:MetaException o2)`
get_databases	`list<string> get databases(1:string pattern) throws(1:MetaException o1)`
get_all_databases	`list<string> get_all_databases() throws(1:MetaException o1)`
get_tables_by_type	`list<string> get_tables_by_type(1: string db_name, 2: string pattern, 3: string tableType) throws (1: MetaException o1)`
get_all_tables	`list<string> get all tables(1: string db name) throws (1: MetaException o1)`
get_table	`Table get_table(1:string dbname, 2:string tbl_name) throws (1:MetaException o1, 2:NoSuchObjectException o2)`
get_tables	`list<string> get_tables(1: string db_name, 2: string pattern) throws (1: MetaException o1)`
get_partitions	`list<Partition> get_partitions(1:string db_name, 2:string tbl_name, 3:i16 max_parts=-1) throws (1:NoSuchObjectException o1, 2:MetaException o2`
get_partition_names	`list<string> get partition names(1:string db name, 2:string tbl_name, 3:i16 max_parts=-1) throws (1:MetaException o2)`

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Microsoft SQL Server 2019

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

8 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
3.1.5 Message Processing Events and Sequencing Rules	Clarified the methods, queries, operations, formats, and parameters that are common to this protocol. For each method, added the request and response body schema definitions to the applicable subsection and moved the request and response body examples to the Protocol Examples section.	Major
3.1.5 Message Processing Events and Sequencing Rules	Moved the request body and response body description and table of parameters to the new Request Body and Response Body section.	Minor
3.1.5.1 Request Body and Response Body	Added section and moved the request body and response body description and table of parameters from the Message Processing Events and Sequencing Rules section.	Minor
3.1.5.2 Payload	Added section.	Major
3.1.5.3 get_all_databases	Moved the table of status codes to the Response Body section.	Minor
3.1.5.3.1 Request Body	Clarified the description of the get_all_databases method and its parameters.	Major
3.1.5.3.2 Response Body	Added the table of status codes from the get_all_databases section.	Minor
3.1.5.4 get_databases	Moved the table of status codes to the Response Body section.	Minor
3.1.5.4.1 Request Body	Clarified the description of the get_databases request body and its parameters.	Major
3.1.5.4.2 Response Body	Added the table of status codes from the get_databases section.	Minor
3.1.5.5 get_database	Moved the table of status codes to the Response Body section.	Minor
3.1.5.5.1 Request Body	Clarified the description of the get_database request body and its parameters.	Major
3.1.5.5.2 Response Body	Added the table of status codes from the get_database section.	Minor

Section	Description	Revision class
3.1.5.6 get_all_tables	Moved the table of status codes to the Response Body section.	Minor
3.1.5.6.1 Request Body	Clarified the description of the get_all_tables request body and its parameters.	Major
3.1.5.6.2 Response Body	Added the table of status codes from the get_all_tables section.	Minor
3.1.5.7 get_tables	Moved the table of status codes to the Response Body section.	Minor
3.1.5.7.1 Request Body	Clarified the description of the get_tables request body and its parameters.	Major
3.1.5.7.2 Response Body	Added the table of status codes from the get_tables section.	Minor
3.1.5.8 get_table	Moved the table of status codes to the Response Body section.	Minor
3.1.5.8.1 Request Body	Clarified the description of the get_table request body and its parameters.	Major
3.1.5.8.2 Response Body	Added the table of status codes from the get_table section.	Minor
3.1.5.9 get_tables_by_type	Moved the table of status codes to the Response Body section.	Minor
3.1.5.9.1 Request Body	Clarified the description of the get_tables_by_type request body and its parameters.	Major
3.1.5.9.2 Response Body	Added the table of status codes from the get_tables_by_type section.	Minor
3.1.5.10 get_partition_names	Moved the table of status codes to the Response Body section.	Minor
3.1.5.10.1 Request Body	Clarified the description of the get_partition_names request body and its parameters.	Major
3.1.5.10.2 Response Body	Added the table of status codes from the get_partition_names section.	Minor
3.1.5.11 get_partitions	Moved the table of status codes to the Response Body section.	Minor
3.1.5.11.1 Request Body	Clarified the description of the get_partitions request body and its parameters.	Major
3.1.5.11.2 Response Body	Added the table of status codes from the get_partitions section.	Minor
4 Protocol Examples	Added and clarified example content that was moved from the Message Processing Events and Sequencing Rules section.	Minor
6.1 Hive_metastore Thrift Schema	Moved the table of API signatures to new Hive_metastore Thrift APIs section.	Minor
6.2 Hive_metastore Thrift APIs	Added new section with table of API signatures moved from the Hive_metastore Thrift Schema section.	Minor

9 Index

A

Applicability 9

C

Capability negotiation 9
Change tracking 65
Common
  Abstract data model 14
  Higher-layer triggered events 14
  Initialization 14
  Message processing events and sequencing rules 14
  Other local events 24
  Timer events 24
  Timers 14

E

Examples
  get_all_databases example 25
  get_all_tables example 27
  get_database example 26
  get_databases example 26
  get_partition_names example 32
  get_partitions example 32
  get_table example 28
  get_tables example 28
  get_tables_by_type example 31

F

Fields - vendor-extensible 9

G

Glossary 6

H

HTTP methods 10

I

Implementer - security considerations 38
Index of security parameters 38
Informative references 7
Introduction 6

M

Messages
  transport 10

N

Namespaces 10
Normative references 7

O

Overview (synopsis) 8

P

Parameters - security index 38
Preconditions 8
Prerequisites 8
Product behavior 64
Protocol Details
  Client 24
  Common 14
Protocol examples
  get_all_databases 25
  get_all_tables 27
  get_database 26
  get_databases 26
  get_partition_names 32
  get_partitions 32
  get_table 28
  get_tables 28
  get_tables_by_type 31

R

References
  informative 7
  normative 7
Relationship to other protocols 8

S

Security
  implementer considerations 38
  parameter index 38
Standards assignments 9

T

Tracking changes 65
Transport 10
  HTTP methods 10
  namespaces 10

V

Vendor-extensible fields 9
Versioning 9