

The workflow developer's guide for the Qserv Ingest system

- 1 Introduction
- 2 What's new in the new Ingest System?
 - What's not done by the new system
- 3 Notes on configuring and running services of the Ingest system
- 4 The version history of the Ingest API
- 5 Error reporting
 - REST services
 - Command-line data loading tools
- 6 An example of a simple workflow
 - An overview of the workflow
 - Qserv setup of the demo
 - The test data set
 - Table schema and the input data
 - Notes on the partitioning parameters
 - Registering a new database in Qserv
 - Registering a table
 - Starting a transaction
 - Allocating chunks
 - Uploading chunk contributions
 - Uploading chunk contributions using the command-line tool
 - What should be done if a contribution upload fails
 - Committing/aborting the transaction
 - Publishing the database
 - Testing ingested catalogs
 - Further steps
- 7 Notes on ingesting regular (fully-replicated) tables
 - Registering regular tables in Qserv
 - Loading data
- 8 Deleting incomplete databases
- 9 Advanced use cases
 - Multiple transactions
 - Parallel ingests
 - Optimizations in using the REST services
 - Building the "director" index
 - Ingesting files directly from workers
 - Ingesting catalogs with many director tables
 - Ingesting RefMatch tables
 - Using MySQL warnings for the data quality control
 - Temporarily un-publishing catalogs to allow adding more tables
 - Concurrency control when processing async requests
 - Specifying character sets when ingesting tables into Qserv
- 10 Information requests
 - About the information requests
 - Status of transactions
 - Databases and database families
 - Key configuration parameters of the Ingest System
- 11 Post-ingest data management and tune-ups
 - Introduction
 - Index management at worker tables
 - Managing statistics for the row counters optimizations
- 12 The API Reference
 - The REST services
 - General guidelines
 - - Calling the REST services
 - - Error reporting
 - - Obtaining the current version of the API
 - - Passing version number with requests
 - Database and table management
 - - Obtaining descriptions of existing databases and database families
 - - Registering databases
 - - Registering tables
 - - Publishing databases
 - - Un-publishing databases to allow adding more tables
 - - Deleting databases and tables
 - Transaction management
 - - Start a transaction
 - - Commit or abort a transaction
 - - Get info on transactions
 - Table location services
 - - Allocate/locate chunks of the partitioned tables
 - - Locate regular tables
 - Information services
 - - Chunk disposition
 - - Status of the contribution requests
 - Director index management

- - (Re-)building indexes
- Worker ingest services
 - - About the services
 - - Ingesting a single file
 - - Re-trying failed contribution
- Ingest configuration
 - - Setting configuration parameters
 - - Retrieving configuration parameters
- Row counters
 - - Collecting row counters and deploying at Qserv
 - - Deleting row counters for a table
 - - Retrieving rows counters of a table
- The command-line tools
 - General guidelines
 - - Error reporting in the command-line tools
 - Contribution loading tools
 - - qserv-replica-file
- 13 APPENDIX
 - Managing indexes of MySQL tables at Qserv workers

1 Introduction

In the original implementation of Qserv, nearly any catalog ingest requirements were mostly driven by the needs of the development and integration tests. These requirements were successfully addressed by a simple tool `qserv-data-loader.py` backed by the Python-based service `wmgr` run at each worker node. ~~This mechanism is still available since it works fairly well for scenarios it was developed for.~~ Unfortunately, it quickly falls short in many areas where ingesting large quantities of data is needed, or where the high performance or reliability of the ingests is mandatory. It's also clear that the simple mechanism won't be able to address the primary target - ingesting the LSST's **Data Release Products**. The number and variety of these cases started growing over the last year as the Project (LSST) was approaching its critical stages, and more complex use case scenarios were showing up for Qserv. Eventually, a need in having a more sophisticated and versatile **Ingest System** emerged.

The current article documents the new system while retaining the main focus on the practical aspects of using the system. More documents on the requirements and the low-level technical details of its implementation (unless it's needed for the purposes of the document's goals) can be found elsewhere.

And the final comment is on the information flow in the document. It's recommended to read the document sequentially. Most ideas presented in the document are introduced in section [An example of a simple workflow](#). The section is followed by a few more sections covering advanced topics. And [The API Reference](#) section at the very end of the document should be used to find complete descriptions of the REST services and tools mentioned in the document.

2 What's *new* in the new Ingest System?

A fundamental difference between the new system from the older one is that it's not just a fixed "loader" tool (backed by a simple set of services). It's rather a set of open interfaces and tools allowing the construction of a variety of ingest *workflows* suitable for specific deployments and use cases of Qserv. In the new paradigm, the older "loader" tool mentioned in the [Introduction](#) section would be just the simplest workflow.

At the very high level, the system is comprised of:

- A REST server that is integrated into the **Master Replication Controller**. It provides a collection of services for managing *metadata* and *states* of the new catalogs to be ingested. The server also coordinates its own operations with **Qserv** and the **Replication System** to prevent interferences with those and minimize failures during catalog ingest activities.
- The **Data Ingest** services run at each Qserv worker alongside the **Replication System**'s worker services. The role of these services is to actually ingest the client's data into the corresponding MySQL tables. The services would also do an additional (albeit, minimal) preprocessing and data transformation (where or when needed) before ingesting the input data into MySQL. Each worker server also includes its own REST server for processing the "by reference" ingest requests as well as various metadata requests in the scope of the workers.

Implementation-wise, the **Ingest System** heavily relies on many functions of the **Replication System** by using many functions of the latter, including the **Replication System's Controller Framework**, various (including the **Configuration**) services, and the worker-side server infrastructure of the **Replication System**.

Client *workflows* interact with the system's services via *open* interfaces (based on the HTTP protocol, REST services, JSON data format, etc.) and use ready-to-use tools to fulfill their goals of ingesting catalogs.

Here is a brief summary of the features of the new system:

- It introduces well-defined semantics into the ingest process. With that, a process of ingesting a new catalog now has to go through a sequence of specific steps maintaining a progressive state of the catalog within Qserv while it's being ingested. The state transitions and the corresponding enforcements made by the system would always ensure that the catalog would be in a well-defined consistent state during each step of the process. Altogether, this model increases the robustness of the process, and it also makes it more efficient.
- To facilitate and implement the above-mentioned *state transitions* the new system introduces a distributed *checkpointing* mechanism named *super-transactions*. These transactions allow for incremental updates of the overall state while allowing to safely *roll back* to a prior *consistent* state should any problem occur during data loading within such transactions.
- In its very foundation, the system has been *designed for constructing high-performance and parallel* ingest workflows w/o compromising the consistency of the ingested catalogs.

- Unlike the original system, the new data loading protocol is *binary*, thus resulting in a more efficient data transfer. To interact with the **Data Ingest** services users may choose either the C++ API or ready-to-use file-loading tools. See [Uploading data using the command-line tool](#) for further details.
- In addition to the above-mentioned binary protocol, the system also supports ingesting contributions "by reference". In this case, the input data will be pulled by the worker services from the remote locations specified by the ingest workflows. The presently supported sources presently include the object stores (via the HTTP/HTTPS protocols) and the locally mounted distributed filesystems (via the POSIX protocol). See [Ingesting files directly from workers](#) for further details.
- The data loading services also collect and retain within the persistent state of the system as much information as possible on various abnormal conditions that may occur during reading, interpreting, or loading the data into Qserv. This information is made available to the ingest workflows for analyzing problems with the input data, data sources, or configurations of the catalogs (including table schemas). To get further info on this subject, see the sections [Error reporting](#) and [Using MySQL warnings for the data quality control](#). In addition, there are many REST services for obtaining metadata on the state of catalogs, tables, distributed transactions, contribution requests, the progress of the requested operations, etc.

What's not done by the new system

As per its current implementation (which may change in the future) the system will **not** automatically partition input files. This task is expected to be the responsibility of the ingest *workflows*.

Also, the system will **not** (with the very small exception of adding an extra leading column `qserv_trans_id` required by the implementation of the new system) pre-process the input `TSV` / `CSV` files sent to the Ingest Data Servers for loading into tables. It's up to the *workflows* to sanitize the input data and to make them ready to be ingested into Qserv.

3 Notes on configuring and running services of the Ingest system

4 The version history of the Ingest API

5 Error reporting

[REST services](#)

[Command-line data loading tools](#)

6 An example of a simple workflow

[An overview of the workflow](#)

[Qserv setup of the demo](#)

[The test data set](#)

[Table schema and the input data](#)

[Notes on the partitioning parameters](#)

[Registering a new database in Qserv](#)

[Registering a table](#)

[Starting a transaction](#)

[Allocating chunks](#)

[Uploading chunk contributions](#)

[Uploading chunk contributions using the command-line tool](#)

[What should be done if a contribution upload fails](#)

[Committing/aborting the transaction](#)

[Publishing the database](#)

Testing ingested catalogs

Further steps

7 Notes on ingesting regular (fully-replicated) tables

Registering regular tables in Qserv

Loading data

8 Deleting incomplete databases

9 Advanced use cases

Multiple transactions

Parallel ingests

Optimizations in using the REST services

Building the "director" index

Ingesting files directly from workers

Ingesting catalogs with many director tables

Ingesting RefMatch tables

Using MySQL warnings for the data quality control

Temporarily un-publishing catalogs to allow adding more tables

Concurrency control when processing async requests

Specifying character sets when ingesting tables into Qserv

10 Information requests

About the information requests

Status of transactions

Databases and database families

Key configuration parameters of the Ingest System

11 Post-ingest data management and tune-ups

Introduction

Index management at worker tables

Managing statistics for the row counters optimizations

12 The API Reference

The REST services

General guidelines

- Calling the REST services
- Error reporting
- Obtaining the current version of the API
- Passing version number with requests

Database and table management

- Obtaining descriptions of existing databases and database families
- Registering databases
- Registering tables
- Publishing databases
- Un-publishing databases to allow adding more tables
- Deleting databases and tables

Transaction management

- Start a transaction
- Commit or abort a transaction
- Get info on transactions

Table location services

- Allocate/locate chunks of the partitioned tables
- Locate regular tables

Information services

- Chunk disposition
- Status of the contribution requests

Director index management

- (Re-)building indexes

Worker ingest services

- About the services
- Ingesting a single file
- Re-trying failed contribution

Ingest configuration

- [Setting configuration parameters](#)
- [Retrieving configuration parameters](#)

[Row counters](#)

- [Collecting row counters and deploying at Qserv](#)
- [Deleting row counters for a table](#)
- [Retrieving rows counters of a table](#)

[The command-line tools](#)

[General guidelines](#)

- [Error reporting in the command-line tools](#)

[Contribution loading tools](#)

- [qserv-replica-file](#)

[13 APPENDIX](#)

[Managing indexes of MySQL tables at Qserv workers](#)