Hazelcast Documentation

version 3.6.2

May 05, 2016

In-Memory Data Grid - Hazelcast | Documentation: version 3.6.2

Publication date May 05, 2016

Contents

1 Document Revision History **5**

2 C++ Client **7**

2.1 Setting Up C++ Client 7

2.2 Installing C++ Client 8

2.3 Compiling C++ Client 8

2.3.1 Linux C++ Client 8

2.3.2 Mac C++ Client 8

2.3.3 Windows C++ Client 8

2.4 Serialization Support 8

2.4.1 Custom Serialization 9

2.5 Raw Pointer API 9

2.6 Query API 11

2.7 C++ Client Code Examples 11

2.7.1 C++ Client Map Example 12

2.7.2 C++ Client Queue Example 12

2.7.3 C++ Client Serialization Example 14

3 Glossary **17**

Chapter 1

Document Revision History

This chapter lists the changes made to this document from the previous release.

NOTE: *Please refer to the Release Notes for the new features, enhancements and fixes performed for each Hazelcast C++ Client release.*

Chapter	Section	Description
Chapter 2 - C++ Client		Added new sections Custom Serialization and Raw Pointer API.

Chapter 2

C++ Client

You can use Native C++ Client to connect to Hazelcast cluster members and perform almost all operations that a member can perform. Clients differ from members in that clients do not hold data. The C++ Client is by default a smart client, i.e., it knows where the data is and asks directly for the correct member. You can disable this feature (using the `ClientConfig::setSmart` method) if you do not want the clients to connect to every member.

The features of C++ Clients are listed below:

- Access to distributed data structures (IMap, IQueue, MultiMap, ITopic, etc.).
- Access to transactional distributed data structures (TransactionalMap, TransactionalQueue, etc.).
- Ability to add cluster listeners to a cluster and entry/item listeners to distributed data structures.
- Distributed synchronization mechanisms with ILock, ISemaphore and ICountDownLatch.

2.1 Setting Up C++ Client

Hazelcast C++ Client is shipped with 32/64 bit, shared and static libraries. You only need to include the boost *shared_ptr.hpp* header in your compilation since the API makes use of the boost `shared_ptr`.

The downloaded release folder consists of:

- Mac_64/
- Windows_32/
- Windows_64/
- Linux_32/
- Linux_64/
- docs/ *(HTML Doxygen documents are here)*

Each of the folders above contains the following:

- examples/ There are a number of examples in this folder for each feature. Each example produces an executable which you can run in a cluster. You may need to set the server IP addresses for the examples to run.
- hazelcast/
  - lib/ => Contains both shared and static library of hazelcast.
  - include/ => Contains headers of client.
- external/
  - include/ => Contains headers of dependencies. (boost::shared_ptr)

2.2 Installing C++ Client

The C++ Client is tested on Linux 32/64-bit, Mac 64-bit and Windows 32/64-bit machines. For each of the headers above, it is assumed that you are in the correct folder for your platform. Folders are Mac_64, Windows_32, Windows_64, Linux_32 or Linux_64.

2.3 Compiling C++ Client

For compilation, you need to include the `hazelcast/include` and `external/include` folders in your distribution. You also need to link your application to the appropriate static or shared library.

2.3.1 Linux C++ Client

For Linux, there are two distributions: 32 bit and 64 bit.

Here is an example script to build with static library:

```
g++ main.cpp -pthread -I./external/include -I./hazelcast/include       ./hazelcast/lib/libHazelcastClien
```

Here is an example script to build with shared library:

```
g++ main.cpp -lpthread -Wl,-no-as-needed -lrt -I./external/include -I./hazelcast/include
-L./hazelcast/lib -lHazelcastClientShared_64
```

2.3.2 Mac C++ Client

For Mac, there is one distribution: 64 bit.

Here is an example script to build with static library:

```
g++ main.cpp -I./external/include -I./hazelcast/include ./hazelcast/lib/libHazelcastClientStatic_64.a
```

Here is an example script to build with shared library:

```
g++ main.cpp -I./external/include -I./hazelcast/include -L./hazelcast/lib -lHazelcastClientShared_64
```

2.3.3 Windows C++ Client

For Windows, there are two distributions; 32 bit and 64 bit. The static library is located in a folder named “static” while the dynamic library(dll) is in the folder named as “shared”.

When compiling for Windows environment the user should specify one of the following flags: HAZELCAST_USE_STATIC: You want the application to use the static Hazelcast library. HAZELCAST_USE_SHARED: You want the application to use the shared Hazelcast library.

2.4 Serialization Support

C++ client supports the following types of object serializations:

- **Built-in primitive types**: Some primitive types have built-in support for serialization. These are `char`, `unsigned char` (`byte`), `bool`, `short`, `int`, `long`, `float`, `double`, `stl string`and vector of these primitive types.
- **IdentifiedDataSerializable**: This interface enables a fast serialization by providing a unique factory and class IDs. It requires the server side class as well.
- **Portable Serialization**: This serialization carries the meta data for the object structure. If server side deserialization is not needed, you do not need to prepare the server side implementation.

- **Custom Serialization**: This serialization allows you to use an external custom serialization, e.g., Google's Protocol Buffers. It provides serialization support without modifying your existing libraries where object classes exist.

2.4.1 Custom Serialization

If all of your classes that need to be serialized are inherited from the same class, you can use an implementation as shown in the example snippet below:

```
class  MyCustomSerializer : public serialization::Serializer<ExampleBaseClass> {
   public:
      void write(serialization::ObjectDataOutput & out, const ExampleBaseClass& object);
      void read(serialization::ObjectDataInput & in, ExampleBaseClass& object);
      int getHazelcastTypeId() const;
    };
```

If your classes are not inherited from the same base class, you can use a serializer class with templates as shown in the example snippet below:

```
template<typename T>
class MyCustomSerializer : public serialization::Serializer<T> {
       public:
         void write(serialization::ObjectDataOutput & out, const T& object) {
                             //.....
         }
         void read(serialization::ObjectDataInput & in, T& object) {
                            //.....
         }
         int getHazelcastTypeId() const {
                            //..
         }
    };
```

Along with your serializer, you should provide the function `getHazelcastTypeId()` with the same namespace to which `ExampleBaseClass` belongs as shown below:

```
int getHazelcastTypeId(const MyClass*);
```

This function should return the same ID with its serializer. This ID is used to determine which serializer needs to be used for your classes.

After you implement your serializer, you can register it using `SerializationConfig` as shown below:

```
clientConfig.getSerializationConfig().
registerSerializer(boost::shared_ptr<hazelcast::client::
serialization::SerializerBase>(new MyCustomSerializer());
```

2.5 Raw Pointer API

When using C++ client you can have the ownership of raw pointers for the objects you create and return. This allows you to keep the objects in your library/application without any need for copy.

For each container you can use the adapter classes, whose names start with `RawPointer`, to access the raw pointers of the created objects. These adapter classes are found in `hazelcast::client::adaptor` namespace and listed below:

- RawPointerList
- RawPointerQueue
- RawPointerTransactionalMultiMap
- RawPointerMap
- RawPointerSet
- RawPointerTransactionalQueue
- RawPointerMultiMap
- RawPointerTransactionalMap

These are adapter classes and they do not create new structures. You just provide the legacy containers as parameters and then you can work with these raw capability containers freely. An example usage of `RawPointerMap` is shown below:

```
hazelcast::client::IMap<std::string, std::string> m = hz.getMap<std::string, std::string>("map");
hazelcast::client::adaptor::RawPointerMap<std::string, std::string> map(m);
map.put("1", "Tokyo");
map.put("2", "Paris");
map.put("3", "New York");
std::cout << "Finished loading map" << std::endl;

std::auto_ptr<hazelcast::client::DataArray<std::string> > vals = map.values();
std::auto_ptr<hazelcast::client::EntryArray<std::string, std::string> > entries = map.entrySet();

std::cout << "There are " << vals->size() << " values in the map" << std::endl;
std::cout << "There are " << entries->size() << " entries in the map" << std::endl;

for (size_t i = 0; i < entries->size(); ++i) {
   const std::string * key = entries->getKey(i);
      if ((std::string *) NULL == key) {
      std::cout << "The key at index " << i << " is NULL" << std::endl;
        } else {
            std::auto_ptr<std::string> val = entries->releaseValue(i);
            std::cout << "(Key, Value) for index " << i << " is: (" << *key << ", " <<
                (val.get() == NULL ? "NULL" : *val) << ")" << std::endl;
        }
    }
    std::cout << "Finished" << std::endl;
```

Raw pointer API uses the DataArray and EntryArray interfaces which allow late deserialization of objects. The entry in the returned array is deserialized only when it is accessed. Please see the example code below:

```
// No deserialization here
std::auto_ptr<hazelcast::client::DataArray<std::string> > vals = map.values();

// deserializes the item at index 0 assuming that there are at least 1 items in the array
const std::string *value = vals->get(0);

// no deserialization here since it was already de-serialized
value = vals->get(0);

// no deserialization here since it was already de-serialized
value = (*vals)[0];

// releases the value so that you can keep this object pointer in your application at some other place
std::auto_ptr<std::string> releasedValue = vals->release(0);

// deserialization occurs again since the value was released already
value = vals->get(0);
```

Using raw pointer based API may improve performance if you are using the API to return multiple values such as values, keySet, and entrySet. In this case, cost of deserialization is delayed until the item is actually accessed.

2.6 Query API

C++ client API allows you to query map values, keys and entries using predicates. It also allows you to use Hazelcast Map's `executeOnKey` and `executeOnEntries` methods with predicates. You can run a processor on a subset of entries with these methods.

You can add entry listeners with predicates using C++ client API. By this way, only the events for the selected subset of entries matching the query criteria are received by your listener.

C++ client API provides a rich set of built-in predicates as supported by the Java client. You can create your own predicates by implementing `Predicate` interfaces both at the C++ client side and server side. Built-in predicates are listed below:

- `AndPredicate`
- `EqualPredicate`
- `ILikePredicate`
- `LikePredicate`
- `OrPredicate`
- `TruePredicate`
- `BetweenPredicate`
- `FalsePredicate`
- `InPredicate`
- `NotEqualPredicate`
- `PagingPredicate`
- `RegexPredicate`
- `GreaterLessPredicate`
- `InstanceOfPredicate`
- `NotPredicate`
- `SqlPredicate`

An example query is shown in the following snippet:

```
IMap<int, int> intMap = client.getMap<int, int>("testValuesWithPredicateIntMap");
adaptor::RawPointerMap<int, int> rawMap(intMap);
// ...
// BetweenPredicate
// 5 <= key <= 10
valuesArray = rawMap.values(query::BetweenPredicate<int>(query::QueryConstants::getKeyAttributeName(), 5
```

This example query returns the values between 5 and 10, inclusive. You can find the examples of each built-in predicate in `distributed-map/query` folder of `examples`.

NOTE: *API that returns pointers may return null pointers for null values. You need to check for null values.*

2.7 C++ Client Code Examples

You can try the following C++ client code examples. You need to have a Hazelcast client member running for the code examples to work.

2.7.1 C++ Client Map Example

```
#include <hazelcast/client/HazelcastAll.h>
#include <iostream>

using namespace hazelcast::client;

int main() {
  ClientConfig clientConfig;
  Address address( "localhost", 5701 );
  clientConfig.addAddress( address );

  HazelcastClient hazelcastClient( clientConfig );

  IMap<int,int> myMap = hazelcastClient.getMap<int ,int>( "myIntMap" );
  myMap.put( 1,3 );
  boost::shared_ptr<int> value = myMap.get( 1 );
  if( value.get() != NULL ) {
    //process the item
  }

  return 0;
}
```

2.7.2 C++ Client Queue Example

```
#include <hazelcast/client/HazelcastAll.h>
#include <iostream>
#include <string>

using namespace hazelcast::client;

int main() {
  ClientConfig clientConfig;
  Address address( "localhost", 5701 );
  clientConfig.addAddress( address );

  HazelcastClient hazelcastClient( clientConfig );

  IQueue<std::string> queue = hazelcastClient.getQueue<std::string>( "q" );
  queue.offer( "sample" );
  boost::shared_ptr<std::string> value = queue.poll();
  if( value.get() != NULL ) {
    //process the item
  }
  return 0;
}
```

2.7.2.1 C++ Client Entry Listener Example

```
#include "hazelcast/client/ClientConfig.h"
#include "hazelcast/client/EntryEvent.h"
#include "hazelcast/client/IMap.h"
#include "hazelcast/client/Address.h"
#include "hazelcast/client/HazelcastClient.h"
#include <iostream>
#include <string>
```

```
using namespace hazelcast::client;

class SampleEntryListener {
  public:

  void entryAdded( EntryEvent<std::string, std::string> &event ) {
    std::cout << "entry added " <<  event.getKey() << " "
        << event.getValue() << std::endl;
  };

  void entryRemoved( EntryEvent<std::string, std::string> &event ) {
    std::cout << "entry added " <<  event.getKey() << " "
        << event.getValue() << std::endl;
  }

  void entryUpdated( EntryEvent<std::string, std::string> &event ) {
    std::cout << "entry added " <<  event.getKey() << " "
        << event.getValue() << std::endl;
  }

  void entryEvicted( EntryEvent<std::string, std::string> &event ) {
    std::cout << "entry added " <<  event.getKey() << " "
        << event.getValue() << std::endl;
  }
};


int main( int argc, char **argv ) {
  ClientConfig clientConfig;
  Address address( "localhost", 5701 );
  clientConfig.addAddress( address );

  HazelcastClient hazelcastClient( clientConfig );

  IMap<std::string,std::string> myMap = hazelcastClient
      .getMap<std::string ,std::string>( "myIntMap" );
  SampleEntryListener *  listener = new SampleEntryListener();

  std::string id = myMap.addEntryListener( *listener, true );
  // Prints entryAdded
  myMap.put( "key1", "value1" );
  // Prints updated
  myMap.put( "key1", "value2" );
  // Prints entryRemoved
  myMap.remove( "key1" );
  // Prints entryEvicted after 1 second
  myMap.put( "key2", "value2", 1000 );

  // WARNING: deleting listener before removing it from Hazelcast leads to crashes.
  myMap.removeEntryListener( id );

  // listen using predicates
  // only listen the events for entries which has the value that matches the
  // string "%VALue%1%", i.e. any string containing the text value1 case insensitive
  id = myMap.addEntryListener(*listener, query::ILikePredicate(
        query::QueryConstants::getValueAttributeName(), "%VALue%1%"), true);
```

```
  // this will generate an event
  myMap.put("key1", "my__value1_new" );

  sleep(1);

  myMap.removeEntryListener( id );

  // Delete listener after removing it from Hazelcast.
  delete listener;
  return 0;
};
```

2.7.3 C++ Client Serialization Example

Assume that you have the following two classes in Java and you want to use them with a C++ client.

```
class Foo implements Serializable {
  private int age;
  private String name;
}

class Bar implements Serializable {
  private float x;
  private float y;
}
```

First, let them implement `Portable` or `IdentifiedDataSerializable` as shown below.

```
class Foo implements Portable {
  private int age;
  private String name;

  public int getFactoryId() {
    // a positive id that you choose
    return 123;
  }

  public int getClassId() {
    // a positive id that you choose
    return 2;
  }

  public void writePortable( PortableWriter writer ) throws IOException {
    writer.writeUTF( "n", name );
    writer.writeInt( "a", age );
  }

  public void readPortable( PortableReader reader ) throws IOException {
    name = reader.readUTF( "n" );
    age = reader.readInt( "a" );
  }
}

class Bar implements IdentifiedDataSerializable {
  private float x;
  private float y;
```

```
  public int getFactoryId() {
    // a positive id that you choose
    return 4;
  }

  public int getId() {
    // a positive id that you choose
    return 5;
  }

  public void writeData( ObjectDataOutput out ) throws IOException {
    out.writeFloat( x );
    out.writeFloat( y );
  }

  public void readData( ObjectDataInput in ) throws IOException {
    x = in.readFloat();
    y = in.readFloat();
  }
}
```

Then, implement the corresponding classes in C++ with same factory and class ID as shown below.

```
class Foo : public Portable {
  public:
  int getFactoryId() const {
    return 123;
  };

  int getClassId() const {
    return 2;
  };

  void writePortable( serialization::PortableWriter &writer ) const {
    writer.writeUTF( "n", name );
    writer.writeInt( "a", age );
  };

  void readPortable( serialization::PortableReader &reader ) {
    name = reader.readUTF( "n" );
    age = reader.readInt( "a" );
  };

  private:
  int age;
  std::string name;
};

class Bar : public IdentifiedDataSerializable {
  public:
  int getFactoryId() const {
    return 4;
  };

  int getClassId() const {
    return 2;
  };
```

```
  void writeData( serialization::ObjectDataOutput& out ) const {
    out.writeFloat(x);
    out.writeFloat(y);
  };

  void readData( serialization::ObjectDataInput& in ) {
    x = in.readFloat();
    y = in.readFloat();
  };

  private:
  float x;
  float y;
};
```

Now, you can use the classes `Foo` and `Bar` in distributed structures. For example, you can use as Key or Value of `IMap` or as an Item in `IQueue`.

Chapter 3

Glossary

Term	Definition
2-phase Commit	2-phase commit protocol is an atomic commitment protocol for distributed systems. It consists of t
ACID	A set of properties (Atomicity, Consistency, Isolation, Durability) guaranteeing that transactions ar
Cache	A high-speed access area that can be either a reserved section of main memory or storage device.
Garbage Collection	Garbage collection is the recovery of storage that is being used by an application when that applica
Hazelcast Cluster	A virtual environment formed by Hazelcast members communicating with each other in the cluster
Hazelcast Partitions	Memory segments containing the data. Hazelcast is built-on the partition concept, it uses partition
IMDG	An in-memory data grid (IMDG) is a data structure that resides entirely in memory, and is distrib
Invalidation	The process of marking an object as being invalid across the distributed cache.
Java heap	Java heap is the space that Java can reserve and use in memory for dynamic memory allocation. A
LRU, LFU	LRU and LFU are two of eviction algorithms. LRU is the abbreviation for Least Recently Used. It
Member	A Hazelcast instance. Depending on your Hazelcast usage, it can refer to a server or a Java virtual
Multicast	A type of communication where data is addressed to a group of destination members simultaneousl
Near Cache	A caching model. When near cache is enabled, an object retrieved from a remote member is put in
NoSQL	"Not Only SQL". A database model that provides a mechanism for storage and retrieval of data th
OSGI	Formerly known as the Open Services Gateway initiative, it describes a modular system and a serv
Race Condition	This condition occurs when two or more threads can access shared data and they try to change it a
RSA	An algorithm developed by Rivest, Shamir and Adleman to generate, encrypt and decrypt keys for
Serialization	Process of converting an object into a stream of bytes in order to store the object or transmit it to
Split Brain	Split brain syndrome, in a clustering context, is a state in which a cluster of members gets divided
Transaction	Means a sequence of information exchange and related work (such as data store updating) that is t