# Using CouchDB

This tutorial will describe the steps required to use the CouchDB as the state database with Hyperledger Fabric. By now, you should be familiar with Fabric concepts and have explored some of the samples and tutorials.

The tutorial will take you through the following steps:

1. [Enable CouchDB in Hyperledger Fabric](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-enable-couch)
2. [Create an index](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-create-index)
3. [Add the index to your chaincode folder](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-add-index)
4. [Install and instantiate the Chaincode](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-install-instantiate)
5. [Query the CouchDB State Database](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-query)
6. [Use best practices for queries and indexes](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-best)
7. [Query the CouchDB State Database With Pagination](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-pagination)
8. [Update an Index](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-update-index)
9. [Delete an Index](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-delete-index)

For a deeper dive into CouchDB refer to [CouchDB as the State Database](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_as_state_database.html) and for more information on the Fabric ledger refer to the [Ledger](https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html) topic. Follow the tutorial below for details on how to leverage CouchDB in your blockchain network.

Throughout this tutorial we will use the [Marbles sample](https://github.com/hyperledger/fabric-samples/blob/master/chaincode/marbles02/go/marbles_chaincode.go) as our use case to demonstrate how to use CouchDB with Fabric and will deploy Marbles to the [Building Your First Network](https://hyperledger-fabric.readthedocs.io/en/release-1.4/build_network.html) (BYFN) tutorial network. You should have completed the task [Install Samples, Binaries and Docker Images](https://hyperledger-fabric.readthedocs.io/en/release-1.4/install.html). However, running the BYFN tutorial is not a prerequisite for this tutorial, instead the necessary commands are provided throughout this tutorial to use the network.

## Why CouchDB?

Fabric supports two types of peer databases. LevelDB is the default state database embedded in the peer node and stores chaincode data as simple key-value pairs and supports key, key range, and composite key queries only. CouchDB is an optional alternate state database that supports rich queries when chaincode data values are modeled as JSON. Rich queries are more flexible and efficient against large indexed data stores, when you want to query the actual data value content rather than the keys. CouchDB is a JSON document datastore rather than a pure key-value store therefore enabling indexing of the contents of the documents in the database.

In [order](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html) to leverage the benefits of CouchDB, namely content-based JSON queries,your data must be modeled in JSON format. You must decide whether to use LevelDB or CouchDB before setting up your network. Switching a peer from using LevelDB to CouchDB is not supported due to data compatibility issues. All peers on the network must use the same database type. If you have a mix of JSON and binary data values, you can still use CouchDB, however the binary values can only be queried based on key, key range, and composite key queries.

## Enable CouchDB in Hyperledger Fabric

CouchDB runs as a separate database process alongside the peer, therefore there are additional considerations in terms of setup, management, and operations. A docker image of [CouchDB](https://hub.docker.com/r/hyperledger/fabric-couchdb/) is available and we recommend that it be run on the same server as the peer. You will need to setup one CouchDB container per peer and update each peer container by changing the configuration found in core.yaml to point to the CouchDB container. The core.yaml file must be located in the directory specified by the environment variable FABRIC\_CFG\_PATH:

* For docker deployments, core.yaml is pre-configured and located in the peer container FABRIC\_CFG\_PATH folder. However when using docker environments, you typically pass environment variables by editing the docker-compose-couch.yaml to override the core.yaml
* For native binary deployments, core.yaml is included with the release artifact distribution.

Edit the stateDatabase section of core.yaml. Specify CouchDB as the stateDatabase and fill in the associated couchDBConfig properties. For more details on configuring CouchDB to work with fabric, refer [here](http://hyperledger-fabric.readthedocs.io/en/master/couchdb_as_state_database.html#couchdb-configuration). To [view](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html) an example of a core.yaml file configured for CouchDB, examine the BYFN docker-compose-couch.yaml in the HyperLedger/fabric-samples/first-network directory.

## Create an index

Why are indexes important?

Indexes allow a database to be queried without having to examine every row with every query, making them run faster and more efficiently. Normally, indexes are built for frequently occurring query criteria allowing the data to be queried more efficiently. To leverage the major benefit of CouchDB – the ability to perform rich queries against JSON data – indexes are not required, but they are strongly recommended for performance. Also, if sorting is required in a query, CouchDB requires an index of the sorted fields.

**Note**

Rich queries that do not have an index will work but may throw a warning in the CouchDB log that the index was not found. However, if a rich query includes a sort specification, then an index on that field is required; otherwise, the query will fail and an error will be thrown.

To demonstrate building an index, we will use the data from the [Marbles sample](https://github.com/hyperledger/fabric-samples/blob/master/chaincode/marbles02/go/marbles_chaincode.go). In this example, the Marbles data structure is defined as:

type marble struct {

ObjectType string `json:"docType"` //docType is used to distinguish the various types of objects in state database

Name string `json:"name"` //the field tags are needed to keep case from bouncing around

Color string `json:"color"`

Size int `json:"size"`

Owner string `json:"owner"`

}

In this structure, the attributes (docType, name, color, size, owner) define the ledger data associated with the asset. The attribute docType is a pattern used in the chaincode to differentiate different data types that may need to be queried separately. When using CouchDB, it recommended to include this docType attribute to distinguish each type of document in the chaincode namespace. (Each chaincode is represented as its own CouchDB database, that is, each chaincode has its own namespace for keys.)

With respect to the Marbles data structure, docType is used to identify that this document/asset is a marble asset. Potentially there could be other documents/assets in the chaincode database. The documents in the database are searchable against all of these attribute values.

When defining an index for use in chaincode queries, each one must be defined in its own text file with the extension \*.json and the index definition must be formatted in the CouchDB index JSON format.

To define an index, three pieces of information are required:

* fields: these are the frequently queried fields
* name: name of the index
* type: always json in this context

For example, a simple index named foo-index for a field named foo.

{

"index": {

"fields": ["foo"]

},

"name" : "foo-index",

"type" : "json"

}

Optionally the design document attribute ddoc can be specified on the index definition. A [design document](http://guide.couchdb.org/draft/design.html) is CouchDB construct designed to contain indexes. Indexes can be grouped into design documents for efficiency but CouchDB recommends one index per design document.

**Tip**

When defining an index it is a good practice to include the ddoc attribute and value along with the index name. It is important to include this attribute to ensure that you can update the index later if needed. Also it gives you the ability to explicitly specify which index to use on a query.

Here is another example of an index definition from the Marbles sample with the index name indexOwner using multiple fields docType and owner and includes the ddoc attribute:

{

"index":{

"fields":["docType","owner"] **//** Names of the fields to be queried

},

"ddoc":"indexOwnerDoc", **//** (optional) Name of the design [document](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html) **in** which the index will be created**.**

"name":"indexOwner",

"type":"json"

}

In the example above, if the design document indexOwnerDoc does not already exist, it is automatically created when the index is deployed. An index can be constructed with one or more attributes specified in the list of fields and any combination of attributes can be specified. An attribute can exist in multiple indexes for the same docType. In the following example, index1 only includes the attribute owner, index2 includes the attributes owner and color and index3 includes the attributes owner, color and size. Also, notice each index definition has its own ddoc value, following the CouchDB recommended practice.

{

"index":{

"fields":["owner"] **//** Names of the fields to be queried

},

"ddoc":"index1Doc", **//** (optional) Name of the design document **in** which the index will be created**.**

"name":"index1",

"type":"json"

}

{

"index":{

"fields":["owner", "color"] **//** Names of the fields to be queried

},

"ddoc":"index2Doc", **//** (optional) Name of the design document **in** which the index will be created**.**

"name":"index2",

"type":"json"

}

{

"index":{

"fields":["owner", "color", "size"] **//** Names of the fields to be queried

},

"ddoc":"index3Doc", **//** (optional) Name of the design document **in** which the index will be created**.**

"name":"index3",

"type":"json"

}

In general, you should model index fields to match the fields that will be used in query filters and sorts. For more details on building an index in JSON format refer to the [CouchDB documentation](http://docs.couchdb.org/en/latest/api/database/find.html" \l "db-index).

A final word on indexing, Fabric takes care of indexing the documents in the database using a pattern called index warming. CouchDB does not typically index new or updated documents until the next query. Fabric ensures that indexes stay ‘warm’ by requesting an index update after every block of data is committed. This ensures queries are fast because they do not have to index documents before running the query. This process keeps the index current and refreshed every time new records are added to the state database.

## Add the index to your chaincode folder

Once you finalize an index, it is ready to be packaged with your chaincode for deployment by being placed alongside it in the appropriate metadata folder.

If your chaincode installation and instantiation uses the Hyperledger Fabric Node SDK, the JSON index files can be located in any folder as long as it conforms to this [directory structure](https://fabric-sdk-node.github.io/tutorial-metadata-chaincode.html). During the chaincode installation using the client.installChaincode() API, include the attribute (metadataPath) in the [installation request](https://fabric-sdk-node.github.io/global.html#ChaincodeInstallRequest). The value of the metadataPath is a string representing the absolute path to the directory structure containing the JSON index file(s).

Alternatively, if you are using the peer-commands to install and instantiate the chaincode, then the JSON index files must be located under the path META-INF/statedb/couchdb/indexes which is located inside the directory where the chaincode resides.

The [Marbles sample](https://github.com/hyperledger/fabric-samples/tree/master/chaincode/marbles02/go) below illustrates how the index is packaged with the chaincode which will be installed using the peer commands.

[](https://hyperledger-fabric.readthedocs.io/en/release-1.4/_images/couchdb_tutorial_pkg_example.png)

This sample includes one index named indexOwnerDoc:

{"index":{"fields":["docType","owner"]},"ddoc":"indexOwnerDoc", "name":"indexOwner","type":"json"}

### Start the network

**Try it yourself**

Before installing and instantiating the marbles chaincode, we need to start up the BYFN network. For the sake of this tutorial, we want to operate from a known initial state. The following command will kill any active or stale docker containers and remove previously generated artifacts. Therefore let’s run the following command to clean up any previous environments:

cd fabric**-**samples**/**first**-**network

**./**byfn**.**sh down

Now start up the BYFN network with CouchDB by running the following command:

**./**byfn**.**sh up **-**c mychannel **-**s couchdb

This will create a simple Fabric network consisting of a single channel named mychannel with two organizations (each maintaining two peer nodes) and an ordering service while using CouchDB as the state database.

## Install and instantiate the Chaincode

Client applications interact with the blockchain ledger through chaincode. As such we need to install the chaincode on every peer that will execute and endorse our transactions and instantiate the chaincode on the channel. In the previous section, we demonstrated how to package the chaincode so they should be ready for deployment.

Chaincode is installed onto a peer and then instantiated onto the channel using peer-commands.

1. Use the [peer chaincode install](http://hyperledger-fabric.readthedocs.io/en/master/commands/peerchaincode.html?%20chaincode%20instantiate#peer-chaincode-install) command to install the Marbles chaincode on a peer.

**Try it yourself**

Assuming you have started the BYFN network, navigate into the CLI container using the command:

docker exec **-**it cli bash

Use the following command to install the Marbles chaincode from the git repository onto a peer in your BYFN network. The CLI container defaults to using peer0 of org1:

peer chaincode install **-**n marbles **-**v 1.0 **-**p github**.**com**/**chaincode**/**marbles02**/**go

2. Issue the [peer chaincode instantiate](http://hyperledger-fabric.readthedocs.io/en/master/commands/peerchaincode.html?%20chaincode%20instantiate#peer-chaincode-instantiate) command to instantiate the chaincode on a channel.

**Try it yourself**

To instantiate the Marbles sample on the BYFN channel mychannel run the following command:

export CHANNEL\_NAME=mychannel

peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n marbles -v 1.0 -c '{"Args":["init"]}' -P "OR ('Org0MSP.peer','Org1MSP.peer')"

### Verify index was deployed

Indexes will be deployed to each peer’s CouchDB state database once the chaincode is both installed on the peer and instantiated on the channel. You can verify that the CouchDB index was created successfully by examining the peer log in the Docker container.

**Try it yourself**

To view the logs in the peer docker container, open a new Terminal window and run the following command to grep for message confirmation that the index was created.

docker logs peer0**.**org1**.**example**.**com 2**>&**1 **|** grep "CouchDB index"

You should see a result that looks like the following:

[couchdb] CreateIndex **->** INFO 0be Created CouchDB index [indexOwner] **in** state database [mychannel\_marbles] using design document [\_design**/**indexOwnerDoc]

**Note**

If Marbles was not installed on the BYFN peer peer0.org1.example.com, you may need to replace it with the name of a different peer where Marbles was installed.

## Query the CouchDB State Database

Now that the index has been defined in the JSON file and deployed alongside the chaincode, chaincode functions can execute JSON queries against the CouchDB state database, and thereby peer commands can invoke the chaincode functions.

Specifying an index name on a query is optional. If not specified, and an index already exists for the fields being queried, the existing index will be automatically used.

**Tip**

It is a good practice to explicitly include an index name on a query using the use\_index keyword. Without it, CouchDB may pick a less optimal index. Also CouchDB may not use an index at all and you may not realize it, at the low volumes during testing. Only upon higher volumes you may realize slow performance because CouchDB is not using an index and you assumed it was.

### Build the query in chaincode

You can perform complex rich queries against the chaincode data values using the CouchDB JSON query language within chaincode. As we explored above, the [marbles02 sample chaincode](https://github.com/hyperledger/fabric-samples/blob/master/chaincode/marbles02/go/marbles_chaincode.go) includes an index and rich queries are defined in the functions - queryMarbles and queryMarblesByOwner:

* **queryMarbles** –

Example of an **ad hoc rich query**. This is a query where a (selector) string can be passed into the function. This query would be useful to client applications that need to dynamically build their own selectors at runtime. For more information on selectors refer to [CouchDB selector syntax](http://docs.couchdb.org/en/latest/api/database/find.html" \l "find-selectors).

* **queryMarblesByOwner** –

Example of a parameterized query where the query logic is baked into the chaincode. In this case the function accepts a single argument, the marble owner. It then queries the state database for JSON documents matching the docType of “marble” and the owner id using the JSON query syntax.

### Run the query using the peer command

In absence of a client application to test rich queries defined in chaincode, peer commands can be used. Peer commands run from the command line inside the docker container. We will customize the [peer chaincode query](http://hyperledger-fabric.readthedocs.io/en/master/commands/peerchaincode.html?%20chaincode%20query#peer-chaincode-query) command to use the Marbles index indexOwner and query for all marbles owned by “tom” using the queryMarbles function.

**Try it yourself**

Before querying the database, we should add some data. Run the following command in the peer container to create a marble owned by “tom”:

peer chaincode invoke -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n marbles -c '{"Args":["initMarble","marble1","blue","35","tom"]}'

After an index has been deployed during chaincode instantiation, it will automatically be utilized by chaincode queries. CouchDB can determine which index to use based on the fields being queried. If an index exists for the query criteria it will be used. However the recommended approach is to specify the use\_index keyword on the query. The peer command below is an example of how to specify the index explicitly in the selector syntax by including the use\_index keyword:

// Rich Query with index name explicitly specified:

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarbles", "{\"selector\":{\"docType\":\"marble\",\"owner\":\"tom\"}, \"use\_index\":[\"\_design/indexOwnerDoc\", \"indexOwner\"]}"]}'

Delving into the query command above, there are three arguments of interest:

* queryMarbles

Name of the function in the Marbles chaincode. Notice a [shim](https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim) shim.ChaincodeStubInterface is used to access and modify the ledger. The getQueryResultForQueryString() passes the queryString to the shim API getQueryResult().

func (t **\***SimpleChaincode) queryMarbles(stub shim**.**ChaincodeStubInterface, args []string) pb**.**Response {

**//** 0

**//** "queryString"

**if** len(args) **<** 1 {

**return** shim**.**Error("Incorrect number of arguments. Expecting 1")

}

queryString :**=** args[0]

queryResults, err :**=** getQueryResultForQueryString(stub, queryString)

**if** err **!=** nil {

**return** shim**.**Error(err**.**Error())

}

**return** shim**.**Success(queryResults)

}

* {"selector":{"docType":"marble","owner":"tom"}

This is an example of an **ad hoc selector** string which finds all documents of type marble where the owner attribute has a value of tom.

* "use\_index":["\_design/indexOwnerDoc", "indexOwner"]

Specifies both the design doc name indexOwnerDoc and index name indexOwner. In this example the selector query explicitly includes the index name, specified by using the use\_index keyword. Recalling the index definition above [Create an index](https://hyperledger-fabric.readthedocs.io/en/release-1.4/couchdb_tutorial.html#cdb-create-index), it contains a design doc, "ddoc":"indexOwnerDoc". With CouchDB, if you plan to explicitly include the index name on the query, then the index definition must include the ddoc value, so it can be referenced with the use\_index keyword.

The query runs successfully and the index is leveraged with the following results:

Query Result: [{"Key":"marble1", "Record":{"color":"blue","docType":"marble","name":"marble1","owner":"tom","size":35}}]

## Use best practices for queries and indexes

Queries that use indexes will complete faster, without having to scan the full database in couchDB. Understanding indexes will allow you to write your queries for better performance and help your application handle larger amounts of data or blocks on your network.

It is also important to plan the indexes you install with your chaincode. You should install only a few indexes per chaincode that support most of your queries. Adding too many indexes, or using an excessive number of fields in an index, will degrade the performance of your network. This is because the indexes are updated after each block is committed. The more indexes need to be updated through “index warming”, the longer it will take for transactions to complete.

The examples in this section will help demonstrate how queries use indexes and what type of queries will have the best performance. Remember the following when writing your queries:

* All fields in the index must also be in the selector or sort sections of your query for the index to be used.
* More complex queries will have a lower performance and will be less likely to use an index.
* You should try to avoid operators that will result in a full table scan or a full index scan such as $or, $in and $regex.

In the previous section of this tutorial, you issued the following query against the marbles chaincode:

// Example one: query fully supported by the index

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarbles", "{\"selector\":{\"docType\":\"marble\",\"owner\":\"tom\"}, \"use\_index\":[\"indexOwnerDoc\", \"indexOwner\"]}"]}'

The marbles chaincode was installed with the indexOwnerDoc index:

{"index":{"fields":["docType","owner"]},"ddoc":"indexOwnerDoc", "name":"indexOwner","type":"json"}

Notice that both the fields in the query, docType and owner, are included in the index, making it a fully supported query. As a result this query will be able to use the data in the index, without having to search the full database. Fully supported queries such as this one will return faster than other queries from your chaincode.

If you add extra fields to the query above, it will still use the index. However, the query will additionally have to scan the indexed data for the extra fields, resulting in a longer response time. As an example, the query below will still use the index, but will take a longer time to return than the previous example.

// Example two: query fully supported by the index with additional data

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarbles", "{\"selector\":{\"docType\":\"marble\",\"owner\":\"tom\",\"color\":\"red\"}, \"use\_index\":[\"/indexOwnerDoc\", \"indexOwner\"]}"]}'

A query that does not include all fields in the index will have to scan the full database instead. For example, the query below searches for the owner, without specifying the the type of item owned. Since the ownerIndexDoc contains both the owner and docType fields, this query will not be able to use the index.

// Example three: query not supported by the index

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarbles", "{\"selector\":{\"owner\":\"tom\"}, \"use\_index\":[\"indexOwnerDoc\", \"indexOwner\"]}"]}'

In general, more complex queries will have a longer response time, and have a lower chance of being supported by an index. Operators such as $or, $in, and $regex will often cause the query to scan the full index or not use the index at all.

As an example, the query below contains an $or term that will search for every marble and every item owned by tom.

// Example four: query with $or supported by the index

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarbles", "{\"selector\":{"\$or\":[{\"docType\:\"marble\"},{\"owner\":\"tom\"}]}, \"use\_index\":[\"indexOwnerDoc\", \"indexOwner\"]}"]}'

This query will still use the index because it searches for fields that are included in indexOwnerDoc. However, the $or condition in the query requires a scan of all the items in the index, resulting in a longer response time.

Below is an example of a complex query that is not supported by the index.

// Example five: Query with $or not supported by the index

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarbles", "{\"selector\":{"\$or\":[{\"docType\":\"marble\",\"owner\":\"tom\"},{"\color\":"\yellow\"}]}, \"use\_index\":[\"indexOwnerDoc\", \"indexOwner\"]}"]}'

The query searches for all marbles owned by tom or any other items that are yellow. This query will not use the index because it will need to search the entire table to meet the $or condition. Depending the amount of data on your ledger, this query will take a long time to respond or may timeout.

While it is important to follow best practices with your queries, using indexes is not a solution for collecting large amounts of data. The blockchain data structure is optimized to validate and confirm transactions and is not suited for data analytics or reporting. If you want to build a dashboard as part of your application or analyze the data from your network, the best practice is to query an off chain database that replicates the data from your peers. This will allow you to understand the data on the blockchain without degrading the performance of your network or disrupting transactions.

You can use block or chaincode events from your application to write transaction data to an off-chain database or analytics engine. For each block received, the block listener application would iterate through the block transactions and build a data store using the key/value writes from each valid transaction’s rwset. The [Peer channel-based event services](https://hyperledger-fabric.readthedocs.io/en/release-1.4/peer_event_services.html) provide replayable events to ensure the integrity of downstream data stores. For an example of how you can use an event listener to write data to an external database, visit the [Off chain data sample](https://github.com/hyperledger/fabric-samples/tree/master/off_chain_data) in the Fabric Samples.

## Query the CouchDB State Database With Pagination

When large result sets are returned by CouchDB queries, a set of APIs is available which can be called by chaincode to paginate the list of results. Pagination provides a mechanism to partition the result set by specifying a pagesize and a start point – a bookmark which indicates where to begin the result set. The client application iteratively invokes the chaincode that executes the query until no more results are returned. For more information refer to this [topic on pagination with CouchDB](http://hyperledger-fabric.readthedocs.io/en/master/couchdb_as_state_database.html#couchdb-pagination).

We will use the [Marbles sample](https://github.com/hyperledger/fabric-samples/blob/master/chaincode/marbles02/go/marbles_chaincode.go) function queryMarblesWithPagination to demonstrate how pagination can be implemented in chaincode and the client application.

* **queryMarblesWithPagination** –

Example of an **ad hoc rich query with pagination**. This is a query where a (selector) string can be passed into the function similar to the above example. In this case, a pageSize is also included with the query as well as a bookmark.

In order to demonstrate pagination, more data is required. This example assumes that you have already added marble1 from above. Run the following commands in the peer container to create four more marbles owned by “tom”, to create a total of five marbles owned by “tom”:

**Try it yourself**

peer chaincode invoke -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n marbles -c '{"Args":["initMarble","marble2","yellow","35","tom"]}'

peer chaincode invoke -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n marbles -c '{"Args":["initMarble","marble3","green","20","tom"]}'

peer chaincode invoke -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n marbles -c '{"Args":["initMarble","marble4","purple","20","tom"]}'

peer chaincode invoke -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL\_NAME -n marbles -c '{"Args":["initMarble","marble5","blue","40","tom"]}'

In addition to the arguments for the query in the previous example, queryMarblesWithPagination adds pagesize and bookmark. PageSize specifies the number of records to return per query. The bookmark is an “anchor” telling couchDB where to begin the page. (Each page of results returns a unique bookmark.)

* queryMarblesWithPagination

Name of the function in the Marbles chaincode. Notice a [shim](https://godoc.org/github.com/hyperledger/fabric/core/chaincode/shim) shim.ChaincodeStubInterface is used to access and modify the ledger. The getQueryResultForQueryStringWithPagination() passes the queryString along

with the pagesize and bookmark to the shim API GetQueryResultWithPagination().

func (t **\***SimpleChaincode) queryMarblesWithPagination(stub shim**.**ChaincodeStubInterface, args []string) pb**.**Response {

**//** 0

**//** "queryString"

**if** len(args) **<** 3 {

**return** shim**.**Error("Incorrect number of arguments. Expecting 3")

}

queryString :**=** args[0]

**//return** type of ParseInt **is** int64

pageSize, err :**=** strconv**.**ParseInt(args[1], 10, 32)

**if** err **!=** nil {

**return** shim**.**Error(err**.**Error())

}

bookmark :**=** args[2]

queryResults, err :**=** getQueryResultForQueryStringWithPagination(stub, queryString, int32(pageSize), bookmark)

**if** err **!=** nil {

**return** shim**.**Error(err**.**Error())

}

**return** shim**.**Success(queryResults)

}

The following example is a peer command which calls queryMarblesWithPagination with a pageSize of 3 and no bookmark specified.

**Tip**

When no bookmark is specified, the query starts with the “first” page of records.

**Try it yourself**

// Rich Query with index name explicitly specified and a page size of 3:

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarblesWithPagination", "{\"selector\":{\"docType\":\"marble\",\"owner\":\"tom\"}, \"use\_index\":[\"\_design/indexOwnerDoc\", \"indexOwner\"]}","3",""]}'

The following response is received (carriage returns added for clarity), three of the five marbles are returned because the pagsize was set to 3:

[{"Key":"marble1", "Record":{"color":"blue","docType":"marble","name":"marble1","owner":"tom","size":35}},

{"Key":"marble2", "Record":{"color":"yellow","docType":"marble","name":"marble2","owner":"tom","size":35}},

{"Key":"marble3", "Record":{"color":"green","docType":"marble","name":"marble3","owner":"tom","size":20}}]

[{"ResponseMetadata":{"RecordsCount":"3",

"Bookmark":"g1AAAABLeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYqz5yYWJeWkGoOkOWDSOSANIFk2iCyIyVySn5uVBQAGEhRz"}}]

**Note**

Bookmarks are uniquely generated by CouchDB for each query and represent a placeholder in the result set. Pass the returned bookmark on the subsequent iteration of the query to retrieve the next set of results.

The following is a peer command to call queryMarblesWithPagination with a pageSize of 3. Notice this time, the query includes the bookmark returned from the previous query.

**Try it yourself**

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarblesWithPagination", "{\"selector\":{\"docType\":\"marble\",\"owner\":\"tom\"}, \"use\_index\":[\"\_design/indexOwnerDoc\", \"indexOwner\"]}","3","g1AAAABLeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYqz5yYWJeWkGoOkOWDSOSANIFk2iCyIyVySn5uVBQAGEhRz"]}'

The following response is received (carriage returns added for clarity). The last two records are retrieved:

[{"Key":"marble4", "Record":{"color":"purple","docType":"marble","name":"marble4","owner":"tom","size":20}},

{"Key":"marble5", "Record":{"color":"blue","docType":"marble","name":"marble5","owner":"tom","size":40}}]

[{"ResponseMetadata":{"RecordsCount":"2",

"Bookmark":"g1AAAABLeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYqz5yYWJeWkmoKkOWDSOSANIFk2iCyIyVySn5uVBQAGYhR1"}}]

The final command is a peer command to call queryMarblesWithPagination with a pageSize of 3 and with the bookmark from the previous query.

**Try it yourself**

peer chaincode query -C $CHANNEL\_NAME -n marbles -c '{"Args":["queryMarblesWithPagination", "{\"selector\":{\"docType\":\"marble\",\"owner\":\"tom\"}, \"use\_index\":[\"\_design/indexOwnerDoc\", \"indexOwner\"]}","3","g1AAAABLeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYqz5yYWJeWkmoKkOWDSOSANIFk2iCyIyVySn5uVBQAGYhR1"]}'

The following response is received (carriage returns added for clarity). No records are returned, indicating that all pages have been retrieved:

[]

[{"ResponseMetadata":{"RecordsCount":"0",

"Bookmark":"g1AAAABLeJzLYWBgYMpgSmHgKy5JLCrJTq2MT8lPzkzJBYqz5yYWJeWkmoKkOWDSOSANIFk2iCyIyVySn5uVBQAGYhR1"}}]

For an example of how a client application can iterate over the result sets using pagination, search for the getQueryResultForQueryStringWithPagination function in the [Marbles sample](https://github.com/hyperledger/fabric-samples/blob/master/chaincode/marbles02/go/marbles_chaincode.go).

## Update an Index

It may be necessary to update an index over time. The same index may exist in subsequent versions of the chaincode that gets installed. In order for an index to be updated, the original index definition must have included the design document ddoc attribute and an index name. To update an index definition, use the same index name but alter the index definition. Simply edit the index JSON file and add or remove fields from the index. Fabric only supports the index type JSON, changing the index type is not supported. The updated index definition gets redeployed to the peer’s state database when the chaincode is installed and instantiated. Changes to the index name or ddoc attributes will result in a new index being created and the original index remains unchanged in CouchDB until it is removed.

**Note**

If the state database has a significant volume of data, it will take some time for the index to be re-built, during which time chaincode invokes that issue queries may fail or timeout.

### Iterating on your index definition

If you have access to your peer’s CouchDB state database in a development environment, you can iteratively test various indexes in support of your chaincode queries. Any changes to chaincode though would require redeployment. Use the [CouchDB Fauxton interface](http://docs.couchdb.org/en/latest/fauxton/index.html) or a command line curl utility to create and update indexes.

**Note**

The Fauxton interface is a web UI for the creation, update, and deployment of indexes to CouchDB. If you want to try out this interface, there is an example of the format of the Fauxton version of the index in Marbles sample. If you have deployed the BYFN network with CouchDB, the Fauxton interface can be loaded by opening a browser and navigating to http://localhost:5984/\_utils.

Alternatively, if you prefer not use the Fauxton UI, the following is an example of a curl command which can be used to create the index on the database mychannel\_marbles:

// Index for docType, owner. // Example curl command line to define index in the CouchDB channel\_chaincode database

curl **-**i **-**X POST **-**H "Content-Type: application/json" **-**d

"{\"index\":{\"fields\":[\"docType\",\"owner\"]},

\"name\":\"indexOwner\",

\"ddoc\":\"indexOwnerDoc\",

\"type\":\"json\"}" http:**//**hostname:port**/**mychannel\_marbles**/**\_index

**Note**

If you are using BYFN configured with CouchDB, replace hostname:port with localhost:5984.

## Delete an Index

Index deletion is not managed by Fabric tooling. If you need to delete an index, manually issue a curl command against the database or delete it using the Fauxton interface.

The format of the curl command to delete an index would be:

curl **-**X DELETE http:**//**localhost:5984**/**{database\_name}**/**\_index**/**{design\_doc}**/**json**/**{index\_name} **-**H "accept: \*/\*" **-**H "Host: localhost:5984"

To delete the index used in this tutorial, the curl command would be:

curl **-**X DELETE http:**//**localhost:5984**/**mychannel\_marbles**/**\_index**/**indexOwnerDoc**/**json**/**indexOwner **-**H "accept: \*/\*" **-**H "Host: localhost:5984"