

# Golang client library tutorial

This section tutorial will guide you through using the most common RPC endpoints with the golang client library.

You will need to [setup dependencies, install, and run celestia-node](#) if you have not already.

## Project setup

To start, add celestia-openrpc as a dependency to your project:

```
bash go
```

```
get
```

```
github.com/celestiaorg/celestia-openrpc go
```

```
get
```

github.com/celestiaorg/celestia-openrpc To use the following methods, you will need the node URL and your auth token. To get your auth token, see this [guide](#) . To run your node without an auth token, you can use the `--rpc.skip-auth` flag when starting your node. This allows you to pass an empty string as your auth token.

The default URL is `http://localhost:26658` . If you would like to use subscription methods, such as `SubscribeHeaders` below, you must use the `ws` protocol in place of `http` : `ws://localhost:26658` .

## Submitting and retrieving blobs

The [blob.Submit](#) method takes a slice of blobs and a gas price, returning the height the blob was successfully posted at.

- The namespace can be generated with `share.NewBlobNamespaceV0`
- .
- The blobs can be generated with `blob.NewBlobV0`
- .
- You can set `blob.DefaultGasPrice()`
- as the gas price to have celestia-node automatically determine an appropriate gas price.

The [blob.GetAll](#) method takes a height and slice of namespaces, returning the slice of blobs found in the given namespaces.

```
go import ( " bytes " " context " " fmt "
```

```
client
```

```
" github.com/celestiaorg/celestia-openrpc " " github.com/celestiaorg/celestia-openrpc/types/blob " "
github.com/celestiaorg/celestia-openrpc/types/share " )
```

```
// SubmitBlob submits a blob containing "Hello, World!" to the 0xDEADBEEF namespace. It uses the default signer on the
running node. func
```

```
SubmitBlob (ctx context.Context, url string , token string ) error { client, err := client. NewClient (ctx, url, token) if err !=
```

```
nil { return err }
```

```
// let's post to 0xDEADBEEF namespace namespace, err := share. NewBlobNamespaceV0 ([] byte { 0x DE , 0x AD , 0x BE ,
0x EF }) if err !=
```

```
nil { return err }
```

```
// create a blob helloWorldBlob, err := blob. NewBlobV0 (namespace, [] byte ( "Hello, World!" )) if err !=
```

```
nil { return err }
```

```
// submit the blob to the network height, err := client.Blob. Submit (ctx, [] * blob.Blob{helloWorldBlob}, blob. DefaultGasPrice
()) if err !=
```

```
nil { return err }
```

```
fmt. Printf ( "Blob was included at height %d\n " , height)
```

```
// fetch the blob back from the network retrievedBlobs, err := client.Blob. GetAll (ctx, height,
[]share.Namespace{namespace}) if err !=
```

```

nil { return err }

fmt. Printf ( "Blobs are equal? %v\n " , bytes. Equal (helloWorldBlob.Commitment, retrievedBlobs[ 0 ].Commitment)) return

nil } import ( " bytes " " context " " fmt "

client

" github.com/celestiaorg/celestia-openrpc " " github.com/celestiaorg/celestia-openrpc/types/blob " "
github.com/celestiaorg/celestia-openrpc/types/share " )

// SubmitBlob submits a blob containing "Hello, World!" to the 0xDEADBEEF namespace. It uses the default signer on the
running node. func

SubmitBlob (ctx context.Context, url string , token string ) error { client, err := client. NewClient (ctx, url, token) if err !=

nil { return err }

// let's post to 0xDEADBEEF namespace namespace, err := share. NewBlobNamespaceV0 ([] byte { 0x DE , 0x AD , 0x BE ,
0x EF }) if err !=

nil { return err }

// create a blob helloWorldBlob, err := blob. NewBlobV0 (namespace, [] byte ( "Hello, World!" )) if err !=

nil { return err }

// submit the blob to the network height, err := client.Blob. Submit (ctx, [] * blob.Blob{helloWorldBlob}, blob. DefaultGasPrice
()) if err !=

nil { return err }

fmt. Printf ( "Blob was included at height %d\n " , height)

// fetch the blob back from the network retrievedBlobs, err := client.Blob. GetAll (ctx, height,
[]share.Namespace{namespace}) if err !=

nil { return err }

fmt. Printf ( "Blobs are equal? %v\n " , bytes. Equal (helloWorldBlob.Commitment, retrievedBlobs[ 0 ].Commitment)) return

nil }

```

## Subscribing to new headers

You can subscribe to new headers using the [header.Subscribe](#) method. This method returns a channel that will receive new headers as they are produced. In this example, we will fetch all blobs at the height of the new header in the 0xDEADBEEF namespace.

```

go // SubscribeHeaders subscribes to new headers and fetches all blobs at the height of the new header in the
0xDEADBEEF namespace. func

SubscribeHeaders (ctx context.Context, url string , token string ) error { client, err := client. NewClient (ctx, url, token) if err !=

nil { return err }

// create a namespace to filter blobs with namespace, err := share. NewBlobNamespaceV0 ([] byte { 0x DE , 0x AD , 0x BE ,
0x EF }) if err !=

nil { return err }

// subscribe to new headers using a <-chan *header.ExtendedHeader channel headerChan, err := client.Header. Subscribe
(ctx) if err !=

nil { return err }

for { select { case header :=

<- headerChan: // fetch all blobs at the height of the new header blobs, err := client.Blob. GetAll (context. TODO (), header.
Height (), []share.Namespace{namespace}) if err !=

nil { fmt. Printf ( "Error fetching blobs: %v\n " , err) }

```

```

fmt. Printf ( "Found %d blobs at height %d in 0xDEADBEEF namespace \n " , len (blobs), header. Height ()) case
<- ctx. Done (): return

nil } } // SubscribeHeaders subscribes to new headers and fetches all blobs at the height of the new header in the
0xDEADBEEF namespace. func

SubscribeHeaders (ctx context.Context, url string , token string ) error { client, err := client. NewClient (ctx, url, token) if err !=
nil { return err }

// create a namespace to filter blobs with namespace, err := share. NewBlobNamespaceV0 ([] byte { 0x DE , 0x AD , 0x BE ,
0x EF }) if err !=

nil { return err }

// subscribe to new headers using a <-chan *header.ExtendedHeader channel headerChan, err := client.Header. Subscribe
(ctx) if err !=

nil { return err }

for { select { case header :=

<- headerChan: // fetch all blobs at the height of the new header blobs, err := client.Blob. GetAll (context. TODO (), header.
Height (), []share.Namespace{namespace}) if err !=

nil { fmt. Printf ( "Error fetching blobs: %v\n " , err) }

fmt. Printf ( "Found %d blobs at height %d in 0xDEADBEEF namespace \n " , len (blobs), header. Height ()) case

<- ctx. Done (): return

nil } } }

```

## Fetching an Extended Data Square (EDS)

You can fetch an [Extended Data Square \(EDS\)](#) using the [share.GetEDS](#) method. This method takes a header and returns the EDS at the given height.

```

go // GetEDS fetches the EDS at the given height. func

GetEDS (ctx context.Context, url string , token string , height uint64 ) ( * rsmt2d.ExtendedDataSquare, error ) { client, err :=
client. NewClient (ctx, url, token) if err !=

nil { return

nil , err }

// First get the header of the block you want to fetch the EDS from header, err := client.Header. GetByHeight (ctx, height) if
err !=

nil { return

nil , err }

// Fetch the EDS return client.Share. GetEDS (ctx, header) } // GetEDS fetches the EDS at the given height. func

GetEDS (ctx context.Context, url string , token string , height uint64 ) ( * rsmt2d.ExtendedDataSquare, error ) { client, err :=
client. NewClient (ctx, url, token) if err !=

nil { return

nil , err }

// First get the header of the block you want to fetch the EDS from header, err := client.Header. GetByHeight (ctx, height) if
err !=

nil { return

nil , err }

// Fetch the EDS return client.Share. GetEDS (ctx, header) }

```

## API documentation

To see the full list of available methods, see the [API documentation](#) . ([ [Edit this page on GitHub](#) ] Last updated: [Previous page](#) [Node RPC CLI tutorial](#) [Next page](#) [Prompt Scavenger](#) []