If you prefer to watch the video, click below.

{% youtube 9VXkxjDIKCg %}

These functions manage and inspect the network connection between the Meteor client and server.

{% apibox "Meteor.status" %}

This method returns the status of the connection between the client and the server. The return value is an object with the following fields:

{% dtdd name:"connected" type:"Boolean" %} True if currently connected to the server. If false, changes and method invocations will be queued up until the connection is reestablished. {% enddtdd %}

{% dtdd name:"status" type:"String" %} Describes the current reconnection status. The possible values are `connected` (the connection is up and running), `connecting` (disconnected and trying to open a new connection), `failed` (permanently failed to connect; e.g., the client and server support different versions of DDP), `waiting` (failed to connect and waiting to try to reconnect) and `offline` (user has disconnected the connection). {% enddtdd %}

{% dtdd name:"retryCount" type:"Number" %} The number of times the client has tried to reconnect since the connection was lost. 0 when connected. {% enddtdd %}

{% dtdd name:"retryTime" type:"Number or undefined" %} The estimated time of the next reconnection attempt. To turn this into an interval until the next reconnection, use `retryTime - (new Date()).getTime()`. This key will be set only when `status` is `waiting`. {% enddtdd %}

{% dtdd name:"reason" type:"String or undefined" %} If `status` is `failed`, a description of why the connection failed. {% enddtdd %}

Instead of using callbacks to notify you on changes, this is a [reactive]() data source. You can use it in a [template]() or [computation]() to get realtime updates.

{% apibox "Meteor.reconnect" %}

{% apibox "Meteor.disconnect" %}

Call this method to disconnect from the server and stop all live data updates. While the client is disconnected it will not receive updates to collections, method calls will be queued until the connection is reestablished, and hot code push will be disabled.

Call [Meteor.reconnect]() to reestablish the connection and resume data transfer.

This can be used to save battery on mobile devices when real time updates are not required.

{% apibox "Meteor.onConnection" %}

`onConnection` returns an object with a single method `stop`. Calling `stop` unregisters the callback, so that this callback will no longer be called on new connections.

The callback is called with a single argument, the server-side `connection` representing the connection from the client. This object contains the following fields:

{% dtdd name:"id" type:"String" %} A globally unique id for this connection. {% enddtdd %}

{% dtdd name:"close" type:"Function" %} Close this DDP connection. The client is free to reconnect, but will receive a different connection with a new `id` if it does. {% enddtdd %}

{% dtdd name:"onClose" type:"Function" %} Register a callback to be called when the connection is closed. If the connection is already closed, the callback will be called immediately. {% enddtdd %}

{% dtdd name:"clientAddress" type:"String" %} The IP address of the client in dotted form (such as `127.0.0.1`).

If you're running your Meteor server behind a proxy (so that clients are connecting to the proxy instead of to your server directly), you'll need to set the `HTTP_FORWARDED_COUNT` environment variable for the correct IP address to be reported by `clientAddress`.

Set `HTTP_FORWARDED_COUNT` to an integer representing the number of proxies in front of your server. For example, you'd set it to `1` when your server was behind one proxy. {% enddtdd %}

{% dtdd name:"httpHeaders" type:"Object" %} When the connection came in over an HTTP transport (such as with Meteor's default SockJS implementation), this field contains whitelisted HTTP headers.

Cookies are deliberately excluded from the headers as they are a security risk for this transport. For details and alternatives, see the [SockJS documentation](). {% enddtdd %}

> *Currently when a client reconnects to the server (such as after temporarily losing its Internet connection), it will get a new connection each time. The `onConnection` callbacks will be called again, and the new connection will have a new connection `id`.*

> *In the future, when client reconnection is fully implemented, reconnecting from the client will reconnect to the same connection on the server: the `onConnection` callback won't be called for that connection again, and the connection will still have the same connection `id`.*

{% apibox "DDP.connect" %}

To call methods on another Meteor application or subscribe to its data sets, call `DDP.connect` with the URL of the application. `DDP.connect` returns an object which provides:

- `subscribe` - Subscribe to a record set. See Meteor.subscribe.
- `call` - Invoke a method. See Meteor.call.
- `apply` - Invoke a method with an argument array. See Meteor.apply.
- `methods` - Define client-only stubs for methods defined on the remote server. See Meteor.methods.
- `status` - Get the current connection status. See Meteor.status.
- `reconnect` - See Meteor.reconnect.
- `disconnect` - See Meteor.disconnect.

By default, clients open a connection to the server from which they're loaded. When you call `Meteor.subscribe`, `Meteor.status`, `Meteor.call`, and `Meteor.apply`, you are using a connection back to that default server.

{% apibox "DDP.onReconnect" %}