Accounts

The Meteor Accounts system builds on top of the `userId` support in `publish` and `methods`. The core packages add the concept of user documents stored in the database, and additional packages add secure password authentication, integration with third party login services, and a pre-built user interface.

The basic Accounts system is in the `accounts-base` package, but applications typically include this automatically by adding one of the login provider packages: `accounts-password`, `accounts-facebook`, `accounts-github`, `accounts-google`, `accounts-meetup`, `accounts-twitter`, or `accounts-weibo`.

Read more about customizing user accounts in the Accounts article in the Meteor Guide.

{% apibox "Meteor.user" %}

Retrieves the user record for the current user from the `Meteor.users` collection.

On the client, the available fields will be those that are published from the server (other fields won't be available on the client). By default the server publishes `username`, `emails`, and `profile` (writable by user). See `Meteor.users` for more on the fields used in user documents.

On the server, this will fetch the record from the database. To improve the latency of a method that uses the user document multiple times, save the returned record to a variable instead of re-calling `Meteor.user()`.

Fetching the full user document can cause unnecessary database usage on the server and over-reactivity on the client, particularly if you store lots of custom data on it. Therefore it is recommended to use the `options` parameter to only fetch the fields you need:

```
const userName = Meteor.user({fields: {'profile.name': 1}}).profile.name;
```

{% apibox "Meteor.userId" %}

{% apibox "Meteor.users" %}

This collection contains one document per registered user. Here's an example user document:

```
{
  _id: 'QwkSmTCZiw5KDx3L6',  // Meteor.userId()
```

```
  username: 'cool_kid_13', // Unique name
  emails: [
    // Each email address can only belong to one user.
    { address: 'cool@example.com', verified: true },
    { address: 'another@different.com', verified: false }
  ],
  createdAt: new Date('Wed Aug 21 2013 15:16:52 GMT-0700 (PDT)'),
  profile: {
    // The profile is writable by the user by default.
    name: 'Joe Schmoe'
  },
  services: {
    facebook: {
      id: '709050', // Facebook ID
      accessToken: 'AAACCgdX7G2...AbV9AZDZD'
    },
    resume: {
      loginTokens: [
        { token: '97e8c205-c7e4-47c9-9bea-8e2ccc0694cd',
          when: 1349761684048 }
      ]
    }
  }
}
```

A user document can contain any data you want to store about a user. Meteor treats the following fields specially:

- `username`: a unique String identifying the user.
- `emails`: an Array of Objects with keys `address` and `verified`; an email address may belong to at most one user. `verified` is a Boolean which is true if the user has verified the address with a token sent over email.
- `createdAt`: the Date at which the user document was created.
- `profile`: an Object which the user can create and update with any data. Do not store anything on `profile` that you wouldn't want the user to edit unless you have a deny rule on the `Meteor.users` collection.
- `services`: an Object containing data used by particular login services. For example, its `reset` field contains tokens used by forgot password links, and its `resume` field contains tokens used to keep you logged in between sessions.

Like all Mongo.Collections, you can access all documents on the server, but only those specifically published by the server are available on the client. You can also use all Collection methods, for instance `Meteor.users.remove` on the server to delete a user.

By default, the current user's `username`, `emails` and `profile` are published to the client. You can publish additional fields for the current user with:

```
// Server
Meteor.publish('userData', function () {
  if (this.userId) {
    return Meteor.users.find({ _id: this.userId }, {
      fields: { other: 1, things: 1 }
    });
  } else {
    this.ready();
  }
});

// Client
Meteor.subscribe('userData');
```

If the autopublish package is installed, information about all users on the system is published to all clients. This includes `username`, `profile`, and any fields in `services` that are meant to be public (eg `services.facebook.id`, `services.twitter.screenName`). Additionally, when using autopublish more information is published for the currently logged in user, including access tokens. This allows making API calls directly from the client for services that allow this.

Users are by default allowed to specify their own `profile` field with `Accounts.createUser` and modify it with `Meteor.users.update`. To allow users to edit additional fields, use `Meteor.users.allow`. To forbid users from making any modifications to their user document:

```
Meteor.users.deny({ update: () => true });
```

{% apibox “Meteor.loggingIn” %}

For example, the `accounts-ui` package uses this to display an animation while the login request is being processed.

{% apibox “Meteor.loggingOut” %}

{% apibox “Meteor.logout” %}

{% apibox “Meteor.logoutOtherClients” %}

For example, when called in a user’s browser, connections in that browser remain logged in, but any other browsers or DDP clients logged in as that user will be logged out.

{% apibox “Meteor.loginWithPassword” %}

If there are multiple users with a username or email only differing in case, a case sensitive match is required. Although `createUser` won’t let you create users with ambiguous usernames or emails, this could happen with existing databases or if you modify the users collection directly.

This method can fail throwing one of the following errors: * “Unrecognized options for login request [400]” if `user` or `password` is undefined. * “Match

failed [400]" if `user` isn't an Object or String, or `password` isn't a String. * "User not found [403]" if the email or username provided in `user` doesn't belong to a registered user. * "Incorrect password [403]" if the password provided is incorrect. * "User has no password set [403]" if `user` doesn't have a password.

This function is provided by the `accounts-password` package. See the Passwords section below.

{% apibox "Meteor.loginWith" %}

Available functions are:

- `Meteor.loginWithMeteorDeveloperAccount`
- `Meteor.loginWithFacebook`
  - `options` may also include Facebook's `auth_type` parameter
- `Meteor.loginWithGithub`
- `Meteor.loginWithGoogle`
  - `options` may also include Google's additional URI parameters
- `Meteor.loginWithMeetup`
- `Meteor.loginWithTwitter`
  - `options` may also include Twitter's `force_login` parameter
- `Meteor.loginWithWeibo`

These functions initiate the login process with an external service (eg: Facebook, Google, etc), using OAuth. When called they open a new pop-up window that loads the provider's login page. Once the user has logged in with the provider, the pop-up window is closed and the Meteor client logs in to the Meteor server with the information provided by the external service.

Requesting Permissions

In addition to identifying the user to your application, some services have APIs that allow you to take action on behalf of the user. To request specific permissions from the user, pass the `requestPermissions` option the login function. This will cause the user to be presented with an additional page in the pop-up dialog to permit access to their data. The user's `accessToken` — with permissions to access the service's API — is stored in the `services` field of the user document. The supported values for `requestPermissions` differ for each login service and are documented on their respective developer sites:

- Facebook: http://developers.facebook.com/docs/authentication/permissions/
- GitHub: http://developer.github.com/v3/oauth/#scopes
- Google: https://developers.google.com/identity/protocols/googlescopes
- Meetup: http://www.meetup.com/meetup_api/auth/#oauth2-scopes
- Twitter, Weibo, Meteor developer accounts: `requestPermissions` currently not supported

External login services typically require registering and configuring your application before use. The easiest way to do this is with the `accounts-ui` package which

presents a step-by-step guide to configuring each service. However, the data can be also be entered manually in the `ServiceConfiguration.configurations` collection, which is exported by the `service-configuration` package.

Configuring Services

First, add the service configuration package:

```
meteor add service-configuration
```

Then, in your app (this example is for the Weebo service):

```
ServiceConfiguration.configurations.upsert(
  { service: 'weibo' },
  {
    $set: {
      loginStyle: "popup",
      clientId: "1292962797", // See table below for correct property name!
      secret: "75a730b58f5691de5522789070c319bc"
    }
  }
);
```

Since Meteor 2.7 you no longer need to manually set the configuration and instead can use Meteor settings by setting your services under `Meteor.settings.packages.service-configuration.<service>`. All the properties can be set under the service and will be added to the database as is, so make sure that they are correct. For the example above, the settings would look like:

```
{
  "packages": {
    "service-configuration": {
      "weibo": {
        "loginStyle": "popup",
        "clientId": "1292962797",
        "secret": "75a730b58f5691de5522789070c319bc"
      }
    }
  }
}
```

The correct property name to use for the API identifier (i.e. `clientId` in the above example) depends on the login service being used, so be sure to use the correct one:

Property Name	Services
`appId`	Facebook

Property Name	Services
`clientId`	Github, Google, Meetup, Meteor Developer Accounts, Weibo
`consumerKey`	Twitter

Additionally, each external service has its own login provider package and login function. For example, to support GitHub login, run the following in your terminal:

```
meteor add accounts-github
```

and use the `Meteor.loginWithGithub` function:

```
Meteor.loginWithGithub({
  requestPermissions: ['user', 'public_repo']
}, (error) => {
  if (error) {
    Session.set('errorMessage', error.reason || 'Unknown error');
  }
});
```

Login service configuration is sent from the server to the client over DDP when your app starts up; you may not call the login function until the configuration is loaded. The function `Accounts.loginServicesConfigured()` is a reactive data source that will return true once the login service is configured; you should not make login buttons visible or active until it is true.

Ensure that your `$ROOT_URL` matches the authorized domain and callback URL that you configure with the external service (for instance, if you are running Meteor behind a proxy server, `$ROOT_URL` should be the externally-accessible URL, not the URL inside your proxy).

Manual settings configuration

You can use `Accounts.loginServiceConfiguration` to view and edit the settings collection:

```
Accounts.loginServiceConfiguration.find();
```

Popup versus redirect flow

When configuring OAuth login with a provider (such as Facebook or Google), Meteor lets you choose a popup- or redirect-based flow. In a popup-based flow, when a user logs in, they will be prompted to login at the provider in a popup window. In a redirect-based flow, the user's whole browser window will be redirected to the login provider, and the window will redirect back to your app when the login is completed.

You can also pick which type of login to do by passing an option to `Meteor.loginWith<ExternalService>`

Usually, the popup-based flow is preferable because the user will not have to reload your whole app at the end of the login flow. However, the popup-based flow requires browser features such as `window.close` and `window.opener` that are not available in all mobile environments. In particular, we recommend using `Meteor.loginWith<ExternalService>({ loginStyle: 'redirect' })` in the following environments:

- Inside UIWebViews (when your app is loaded inside a mobile app)
- In Safari on iOS8 (`window.close` is not supported due to a bug)

{% apibox "currentUser" %}

{% apibox "loggingIn" %}

{% apibox "Accounts.ui.config" %}

Example:

```
Accounts.ui.config({
  requestPermissions: {
    facebook: ['user_likes'],
    github: ['user', 'repo']
  },
  requestOfflineToken: {
    google: true
  },
  passwordSignupFields: 'USERNAME_AND_OPTIONAL_EMAIL'
});
```

Since Meteor 2.7 you can configure these in your Meteor settings under `Meteor.settings.public.packages.accounts-ui-unstyled`.