Adding a new service

Summary

You can take advantage of the pluggable architecture of this library by choosing different implementations for services, and/or adding new service roles altogether. A service is just a Javascript class that inherits from eitherPublicService ,PrivateService , orLocalService , and contains public method(s).

It can depend on other services through our built-independency injection framework , and can also be configured through the Maker config file / config options.

Steps to add a new service

The information below is written for developers who want to add to dai.js, but you can also add services through the use ofplugins . Here are the steps to add a new service called ExampleService:

(1) In the src directory, create an ExampleService.js file in one of the subdirectories.

(2) In src/config/DefaultServiceProvider.js, import ExampleService.js and add it to the _services array

(3) Create a class called ExampleService in ExampleService.js

(4) The service must extend one of:

- PrivateService - requires both a network connection and authentication
- PublicService - requires just a network connection
- LocalService - requires neither
-

See the Service Lifecycle section below for more info

(5) In the constructor, call the parent class's constructor with the following arguments:

- The name of the service. This is how the service can be referenced by other services
- An array of the names of services to depend on
-

(6) Add the necessary public methods

```

Copy //example code in ExampleService.js for steps 3-6 importPublicServicefrom'../core/PublicService';

exportdefaultclassExampleServiceextendsPublicService{ constructor(name='example') { super(name,['log']); }

test(){ this.get('log').info('test'); }

```

(7) If your service will be used to replace a default service (the full list of default service roles can be found in src/config/ConfigFactory), then skip this step. Otherwise, you'll need to add your new service role (e.g. "example") to the ServiceRoles array in src/config/ConfigFactory.

(8) Create a corresponding ExampleService.spec.js file in the test directory. Write a test in the test file that creates a Maker object using your service.

```

Copy //example code in ExampleService.spec.js for step 8 importMakerfrom'../../src/index';

//step 8: a new service role ('example') is used test('test 1',async()=>{ constmaker=awaitMaker.create('http', {example:"ExampleService"}); constexampleService=customMaker.service('example'); exampleService.test();//logs "test" });

//step 8: a custom service replaces a default service (Web3) test('test 2',async()=>{ constmaker=awaitMaker.create('http', {web3:"MyCustomWeb3Service"}); constmycustomWeb3Service=maker.service('web3'); });

```

(9) (Optional) Implement the relevant service lifecycle functions (initialize(), connect(), and authenticate()). See the Service Lifecycle section below for more info

(10) (Optional) Allow for configuration. Service-specific settings can be passed into a service by the Maker config file or config options. These service-specific settings can then be accessed from inside a service as the parameter passed into the initialize function (see the Service Lifecycle section below)

```

Copy //step 10: in ExampleService.spec.js constmaker=awaitMaker.create('http',{ example:["ExampleService",{ exampleSetting:"this is a configuration setting" }] });

//step 10: accessing configuration settings in ExampleService.js initialize(settings) { if(settings.exampleSetting){ this.get('log').info(settings.exampleSetting); } }

```

Service Lifecycle

The three kinds of services mentioned in step 4 above follow the following state machine diagrams in the picture below.

To specify what initializing, connecting and authenticating entails, implement the initialize(), connect(), and authenticate() functions in the service itself. This will be called while the service's manager brings the service to the corresponding state.

```

Copy //example initialize() function in ExampleService.js initialize(settings) { this.get('log').info('ExampleService is initializing...'); this._setSettings(settings); }

```

A service will not finish initializing/connecting/authenticating until all of its dependent services have completed the same state (if applicable - for example a LocalService is considered authenticated/connected in addition to initialized, if it has finished initializing). The example code here shows how to wait for the service to be in a certain state.

```

Copy constmaker=awaitMaker.create('http',{example:"ExampleService"}); constexampleService=customMaker.service('example');

//wait for example service and its dependencies to initialize awaitexampleService.manager().initialize();

//wait for example service and its dependencies to connect awaitexampleService.manager().connect();

//wait for example service and its dependencies to authenticate awaitexampleService.manager().authenticate();

//can also use callback syntax exampleService.manager().onConnected(()=>{ /*executed after connected*/ });

//wait for all services used by the maker object to authenticate maker.authenticate();

```

Adding Custom Events

One way to add an event is to “register” a function that gets called on each new block, using the event service's registerPollEvents() function. For example, here is some code from the price service. this.getEthPrice() will be called on each new block, and if the state has changed from the last call, a price/ETH_USD event will be emitted with the payload { price: [new_price] }.

Another way to an add an event is to manually emit an event using the event service's emit function. For example, when the Web3Service initializes, it emits an event that contains info about the provider.

Note that calling registerPollEvents and emit() directly on the event service like in the previous two examples will register events on the "default" event emitter instance. However, you can create a new event emitter instance for your new service. For example, the CDP object defines it's own event emitter, as can be seen here, by calling the event service's buildEmitter() function.

```

Copy //in PriceService.js this.get('event').registerPollEvents({ 'price/ETH_USD':{ price:()=>this.getEthPrice() } });

//in Web3Service.js this.get('event').emit('web3/INITIALIZED',{ provider:{...settings.provider } });

//in the constructor in the Cdp.js this._emitterInstance=this._cdpService.get('event').buildEmitter(); this.on=this._emitterInstance.on; this._emitterInstance.registerPollEvents({ COLLATERAL:{ USD:
```

()=>this.getCollateralValueInUSD(), ETH:()=>this.getCollateralValueInEth() }, DEBT:{ dai:()=>this.getDebtValueInDai() } });

```

Previous Using multiple accounts Next Single-Collateral Sai Last updated4 years ago On this page *Summary * Steps to add a new service * Service Lifecycle * Adding Custom Events

Export as PDF
```