A ReactiveDict stores an arbitrary set of key-value pairs. Use it to manage internal state in your components, ie. like the currently selected item in a list. Each key is individully reactive such that calling `set` for a key will invalidate any Computations that called `get` with that key, according to the usual contract for reactive data sources.

That means if you call [`ReactiveDict#get`]`('currentList')` from inside a Blaze template helper, the template will automatically be rerendered whenever [`ReactiveDict#set`]`('currentList', x)` is called.

To use `ReactiveDict`, add the `reactive-dict` package to your project by running in your terminal:

```
meteor add reactive-dict
```

{% apibox "ReactiveDict" %}

If you provide a name to its constructor, its contents will be saved across Hot Code Push client code updates.

{% apibox "ReactiveDict#set" %}

Example:

```
const state = new ReactiveDict();
state.set('currentRoomId', 'random')

Tracker.autorun(() => {
  Meteor.subscribe('chatHistory', { room: state.get('currentRoomId') });
});

// Causes the function passed to `Tracker.autorun` to be rerun, so that the
// 'chatHistory' subscription is moved to the room 'general'.
state.set('currentRoomId', 'general');
```

`ReactiveDict.set` can also be called with an object of keys and values, which is equivalent to calling `ReactiveDict.set` individually on each key/value pair.

```
const state = new ReactiveDict();
state.set({
  a: 'foo',
  b: 'bar'
});
```

{% apibox "ReactiveDict#setDefault" %}

This is useful in initialization code, to avoid re-initializing your state every time a new version of your app is loaded.

{% apibox "ReactiveDict#get" %}

Example:

```
<!-- main.html -->
<template name="main">
  <p>We've always been at war with {{theEnemy}}.</p>
  <button class="change-enemy">Change Enemy</button>
</template>
```

```
// main.js
Template.main.onCreated(function () {
    this.state = new ReactiveDict();
    this.state.set('enemy', 'Eastasia');
});
Template.main.helpers({
  theEnemy() {
    const inst = Template.instance();
    return inst.state.get('enemy');
  }
});
Template.main.events({
  'click .change-enemy'(event, inst) {
    inst.state.set('enemy', 'Eurasia')
  }
});

// Clicking the button will change the page to say "We've always been at war with
Eurasia"
```

{% apibox "ReactiveDict#equals" %}

If value is a scalar, then these two expressions do the same thing:

```
const state = new ReactiveDict()
// ...
state.get('key') === value
state.equals('key', value)
```

However, the second is recommended, as it triggers fewer invalidations (template redraws), making your program more efficient.

{% apibox "ReactiveDict#all" %}

{% apibox "ReactiveDict#clear" %}

{% apibox "ReactiveDict#destroy" %}