Tutorial: Message Passing

Edit New Page

Michael Thuy edited this page on 30 May 2015 · 4 revisions

In this section, we'll continue the Counter demo and define a message-passing interface between our HTML page and the backend root module. The root module will accept user click actions from the HTML page, keep a global count, and return the count back to the user.

The source code for this tutorial can be found here:
Repository-01

screenshot

> Note. Most tutorials are not yet completed. While we work on these tutorials, please feel free to email us.

TO COMPLETE. REST IS JUNK

```
HIGH-LEVEL DIAGRAM HERE
index.html <=> manifest.json
```

Now that we have included *freedom.js* in our page, a global **freedom** object is exposed in both `index.html` and `counter.js`. In this tutorial we'll use freedom.on and freedom.emit.

One thing to keep in mind during development is that *freedom.js* modules can reside locally or remotely. In fact, a *freedom.js* app can consist of multiple modules, each of which runs in a different physical location. Thus, one should always design their module interfaces with the understanding that messages might have to pass across the Internet before arriving at their destination. This also means that all modules have their own namespace and operate in independent threads of execution.

In this example, we'll pass the entire document after every key press. Yea, it's inefficient, but you should be smarter in your apps.

Sending the text to the root module from index.html

Let's create a `textarea` tag in `index.html` and listen for key press events. When any key is pressed in this element, we'll send a message to our backend root module with all the contents using freedom.emit. `freedom.emit` takes 2 arguments, the message type and the message contents. When called, it will send all messages to the root module.

```
<html>
...
  <script type='text/javascript'>
  window.onload = function() {
    console.log("Window Loaded");
    var input = document.getElementById('text-input');
    input.onkeydown = function(evt) {
      window.freedom.emit('update-text', input.value);
    };
  };
  </script>
</head>
<body>
...
<textarea id='text-input' rows="20" class='span10'></textarea></div>
...
</html>
```

Pages 36

Find a Page…

- Home
- Applications
- Common Issues
- Compiling freedom.js
- Creating a Social Provider
- Debugging
- Debugging Script Parse Errors
- Existing API Providers
- FAQ
- freedom.js OAuth
- freedom.js structure
- freedom.js structure: Consumer Interface
- freedom.js structure: module cleanup
- freedom.js structure: Module Environment
- freedom.js structure: Provider Interface

Show 21 more pages…

Clone this wiki locally

https://github.com/freedc

Adding a backend module interface to main.js

Now in our `main.js` script, we can call freedom.on to listen for messages of a particular *type*.

```
var document;
freedom.on("update-text", function(text) {
  var result = text.toUpperCase();
  document = result;
  freedom.emit("display-text", result);
});
```

In this example, we listen for "update-text" messages. When `index.html` sends a message with that type, we call the provided callback with the message as an argument. In the function, we'll capitalize the letters, store it in a global variable `document` (scoped to the module's namespace). We then return the result back to `index.html`. Note that when `freedom.emit` is called from the module, messages are directed to the outer encompassing page.

Display the uppercased text in index.html

Now we can similarly setup a `freedom.on` callback in `index.html` to listen for "display-text" messages from `index.html` and update our UI accordingly.

```
freedom.on('display-text', function(data) {
  document.getElementById('text-input').value = data;
});
```

Run it!

Run the demo from the base directory in `freedom.js`:

```
LOCAL=yes make demo
```

Then navigate to **http://localhost:8000/demo/texteditor/01** on your browser. This will compile *freedom.js* and start a local web server using 'python -mSimpleHTTPServer'. If you want to do this yourself, remember to point your `index.html` script tags to the right place.

Debuggin

Next Page

2. Module Dependencies - Adding a child module