Most programming languages have a way of including the code in file A from inside file B so that you can split your code into multiple files. C and C++ have #include; Python has import; Ruby and PHP have require. Some languages like C# do this kind of cross-file communication implicitly at compile time.

For most of its life, the JavaScript language didn’t have an official way of doing this. To solve this problem, people built things that concatenated JavaScript files into one file or built dependency loaders like RequireJS. A lot of web developers simply fill their webpages with <script> tags.

Node wanted to solve this problem elegantly, and its developers implemented a standard module system called *CommonJS*. At its core, CommonJS lets you include code from one file in another.

There are three major components to this module system: requiring built-in modules,

requiring third-party modules, and making your own modules. Let’s see how

they work.

***1: Requiring built-in modules***

Node has a number of built-in modules, ranging from filesystem access in a module

called fs to utility functions in a built-in module called util.

You’ll use Node’s require function to use the url module. require is similar to

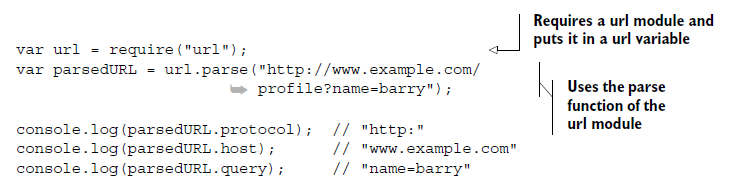
keywords like import or include in other languages. require takes the name of a

package as a string argument and returns a package. There’s nothing special about

the object that’s returned—it’s often an object, but it could be a function or a string

or a number. The next listing shows how you might use the url module.

Example: Requiring Node’s url module:



Most of the time when you’re requiring a module, you’ll put in a variable that has

the same name as the module itself. The previous example puts the url module in a variable of the same name: url.

console.log(urlStr.hostname) ;

console.log(urlStr.protocol) ;

console.log(urlStr.port)

console.log(urlStr.pathname);

console.log(urlStr.query);

https://nodejs.org/api/url.html#url\_url\_pathname